Especial
Delphi e Twitter

Integre suas aplicações Delphi com o maior microblog do mundo

Crie aplicativos personalizados usando a API do Twitter

Clube Delphi +PHP

DevMedia group

Ano 9 :: Edição 111 :: R$ 11,90

DELPHI 2010

Gestures e Multi-Touch

Qualidade de Software
Testes Unitários e Cobertura de Testes

Quick Update
RIA com DataSnap 2010 - REST, JSON, AJAX e Cloud Computing

Boas Práticas
Conheça a nova RTTI do Delphi 2010

Web/ Novidades
Curso de Delphi Prism - Parte 4

Easy Delphi
Transforme classes em componentes e tire o máximo da produtividade RAD do Delphi, usando polimorfismo e abstração

PHP
Smarty Templates – Separe a interface gráfica da lógica de programação no PHP e no Delphi for PHP

ISSN 1517990-7

na web ::

Confira no portal ClubeDelphi PLUS 4 vídeo-aulas:

- Novidades do IDE do Delphi 2010
- Matriz de Rastreabilidade na prática
- Introdução à Matriz de Rastreabilidade
- Criando um KeyLogger

local
Data Center

Nossos **servidores** dedicados possuem monitoramento 24x7x365, link dedicado e **full duplex**, ou seja, os servidores atingem a **velocidade** do link tanto para **upload** quanto para **download.**

HOST

Servidor
Sites
Gerenciamento

DATACENTER

Co-location
E-mail
Backup

TELECOM

Cross Connection
Banda IP
Link PAP

SOLUÇÕES INTELIGENTES

Solicite um
representante comercial

(21)3879-5425
comercial@localdatacenter.com.br

www.localdatacenter.com.br

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Quick Update

6 – RIA com DataSnap 2010
[*Guinther Pauli*]

Delphi

Novidades

8 – Delphi 2010
Novas units, Gestures e Multi-Touch
[*Rodrigo Carreiro*]

Web, Novidades

15 – Curso de Delphi Prism – Parte 4
Melhorias na linguagem – Arrays, Coleções, Loops, Case e Try
[*Guinther Pauli*]

Boa Ideia

21 – Programando com a API do Twitter
Integre suas aplicações Delphi com o microblog mais popular do mundo
[*Daniel Laporte*]

Eng. de Software

Engenharia de Software

27 – Você testa seus aplicativos de forma completa?
Veja como aplicar em Delphi a prática de Code Coverage para melhorar seus testes
[*Paulo Quicoli*]

Boas Práticas

Boas Práticas, Novidades

34 – RTTI
Conheça as novidades na RTTI do Delphi 2010
[*Rodrigo Carreiro*]

Easy

Easy Delphi, Boas Práticas, Tutorial

39 – Introdução à POO – Parte 4
Aprenda de uma forma simples a programar da forma correta
[*Guinther Pauli*]

PHP

Delphi for PHP

46 – Smarty Template
Separe a interface gráfica da lógica de programação
[*Rafael Silva*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Quick Update] Artigos que atualizam o desenvolvedor sobre determinada tecnologia, seja ela nova ou não.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte o dia-a-dia do desenvolvedor mas que prometem ser a próxima febre.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Boa Ideia] Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo é tão criativo que é considerado uma 'boa ideia', dentro do contexto de desenvolvimento de software.

[Tutorial] Artigo no estilo tutorial passo a passo.

[Boas Práticas] Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas mas também as melhores práticas. Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Engenharia de Software] Artigo dentro do contexto de Engenharia de Software.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[Delphi for PHP] Artigo com foco na versão Delphi for PHP.

Clube Delphi +PHP

Ano 9 - 111ª Edição 2009 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Sobrinho
daniel.sobrinho@devmedia.com.br

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

Diagramação e Capa
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Cristiany Queiroz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

Esta edição inaugura nossa cobertura sobre as novidades do Delphi 2010. Começamos com a principal, que provavelmente causará mais impacto nos seus aplicativos: seu usuário poderá agora interagir com suas aplicações usando os dedos! Isso mesmo, é o que chamamos de multi-touch e gestures. Fazendo gestos ou combinações de movimentos com toques na tela o usuário pode acionar actions pré-definidas e acionar comandos da aplicação. Já viu filmes com Minority Report? A Ilha? Conhece a tecnologia Surface da Microsoft? Pessoas interagem com aplicativos fazendo gestos em telas, mesas, arrastando objetos. Aqui tem um vídeo bem interessante de uma aplicação feita em Delphi 2010 que vai ajudar você a ter uma ideia do que estou falando: http://bit.ly/Jjy6O. Que tal chamar o Next do ClientDataSet arrastando o dedo do lado esquerdo da tela até o lado direito? Confira isso e muito mais no artigo do Rodrigo. Ainda sobre as novidades do Delphi 2010, a RTTI (Runtime Type Information) sofreu um grande avanço e está repleta de novidades. O Rodrigo apresenta todas essas novidades, inclusive o suporte ao mais novo recurso da linguagem, já presente em aplicações .NET, agora disponível no Win32, o uso de atributos.

E para não perder o embalo, o Quick Update também comenta um pouco sobre as novidades do Delphi 2010. RIA – Rich Internet Application – é o assunto do momento. Pessoalmente, acredito que a interligação de tecnologias como ADO.NET Data Services, DataSnap 2010, REST, JSON, ATOM/XML com Silverlight e AJAX, além do Cloud Computing, vai representar uma revolução imensa, se já não representa, na forma como vamos trabalhar nossos aplicativos nos próximos anos.

Twitter: Por que daríamos especial destaque nesta edição a essa mania que virou uma febre na Internet, se isto aqui é uma revista de Delphi? Primeiro, o Twitter tem uma API, que pode ser usada para manipular suas funcionalidades a partir de praticamente qualquer linguagem de programação, incluindo obviamente Delphi. Centenas, talvez milhares de aplicações hoje, são construídas para incrementar o Twitter. E deixando um pouco a programação de lado, vamos pensar no Twitter como negócio: empresas estão usando o microblog para manter contato constante com seus clientes, divulgar sua marca, novos produtos, serviços. Que tal sua aplicação ter a capacidade de automaticamente saber o que seus clientes (ou os clientes dos seus clientes) querem, precisam, estão pensando? Feedback imediato? Todos os dias empresas aderem ao serviço, nós não vamos ficar de fora, ainda mais que temos uma vantagem perante os usuários: nós somos programadores!

E muito mais nesta edição! Delphi Prism (D#), Orientação a Objetos, Polimorfismo, Testes Unitários e Cobertura de Testes, Smarty Templates no PHP e muito mais.

Boa leitura e sucesso com o novíssimo Delphi!

Guinther Pauli
guinther@devmedia.com.br
Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix
MCP,MCAD,MCSD.NET,MCTS,MCPD
guintherpauli.blogspot.com
twitter.com/guintherpauli

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

ClubeDelphi PLUS

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+ 700 vídeo aulas e 19 cursos online

Brinde na web desta edição - 4 vídeos! Confira:

4 Vídeos

- Novidades no Delphi 2010 - Novos recursos do IDE: Acompanhe esta série sobre a mais nova versão do Delphi, o Delphi 2010. A partir deste primeiro vídeo veremos: novos recursos do IDE, novas funcionalidades, componentes para trabalhar com Touch, novas propriedades. Muitas melhorias na RTL e VCL. Suporte a atributos, melhorias em Generics e muito mais.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=14082

- Aprimorando a segurança com KeyLogger: Crie junto com Guinther Pauli um aplicativo KeyLogger.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=14065

- Introdução à Matriz de Rastreabilidade – Definições Iniciais: Entenda neste vídeo os conceitos envolvidos para se construir uma matriz de rastreabilidade
http://www.devmedia.com.br/articles/viewcomp.asp?comp=13641

- Introdução à Matriz de Rastreabilidade – Importância e Aplicação: Veja como colocar em prática uma matriz de rastreabilidade
http://www.devmedia.com.br/articles/viewcomp.asp?comp=13642

Para acessar os vídeos utilize os seguintes dados
Login: **dvm.cd** *Senha:* **gesture**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site da DevMedia!
Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **dvm.cd** Senha: **gesture**

Quick Update:

RIA com DataSnap 2010

Guinther Pauli

guinther.pauli@gmail.com

Tem 22 anos de experiência na área de desenvolvimento e arquitetura de software, dos quais 14 dedicados ao Delphi e 9 ao .NET/C#. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET, VB.NET, MSF, SQL Server). Editor geral da Revista ClubeDelphi, Editor Geral da Revista .NET Magazine, professor de Pós-Graduação. Publicou mais de 150 artigos, 400 vídeo-aulas, palestrante em 5 edições da Borland Conference. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas. Blog: guintherpauli.blogspot.com

Muitas empresas e desenvolvedores diariamente me perguntam quais tecnologias serão utilizadas nos próximos anos. Em qual tecnologia devem apostar em um mercado em constantes mudanças, onde tecnologias e siglas para elas surgem a cada minuto e um padrão não é o que o próprio nome sugere. Aqui vai a resposta. Primeiro, os conceitos.

Certamente você já criou alguma aplicação Web e por mais simples que seja, quando o usuário a acessa, há uma requisição ao servidor. Tecnologias Web servidoras tradicionais retornam, além dos dados requisitados, todas as outras informações para apresentação, como cor, fonte, imagens, tabelas utilizadas no layout, entre outros elementos HTML. Ou seja, a apresentação é enviada junto com os dados. Uma arquitetura que está ganhando cada vez mais espaço na Web é a separação dos dados da apresentação. Com a utilização do AJAX os dados podem ser apresentados de forma assíncrona e o servidor não precisa necessariamente mandar a informação já formatada, podendo mandar apenas os dados estruturados. Neste caso a formatação é aplicada usando-se JavaScript, já no cliente. O mesmo se aplica quando pensamos em fazer uma aplicação em Silverlight (tecnologia do .NET para criação de aplicações ricas para internet), que pode solicitar os dados ao servidor e apresentá-los no seu padrão já definido. Neste caso, o cliente faz uma requisição ao servidor, que enviará somente os dados inerentes ao sistema, sem a necessidade de nenhuma informação extra de apresentação, pois o layout já foi definido na construção do aplicativo. Ou seja, uma vez que os dados são recebidos, eles podem ser apresentados em um padrão já definido, seja em Silverlight, Flash, HTML com JavaScript, Desktop, Mobile entre outras tecnologias.

JavaScript Object Notation - JSON - é um modo de estruturar dados utilizando a sintaxe JavaScript. É uma alternativa para o retorno do resultado de consultas REST (visto a seguir). Com JavaScript a grande vantagem de JSON sobre a XML é que não necessita de um manipulador XML (leia-se parser), pois já está estruturado na linguagem, além de sua estrutura ser menor (o que é um ponto positivo a ser considerado quando se fala em tráfego Web). Logo, aplicações AJAX ficam mais simples de serem desenvolvidas. Grandes empresas como Google, Yahoo entre diversas outras, utilizam este formato para serviços Web. REST significa Representational State Transfer. É um conjunto de diretrizes de desenvolvimento de software para distribuição de sistemas que trabalham em cima

do HTTP. Através de requisições HTTP torna possível a troca de informações, através da URL. Estes dados podem ser representados por JSON ou XML. Quando os protocolos de comunicação foram surgindo, como por exemplo, CORBA e SOAP, um movimento contra estes protocolos surgiu, alegando que eles não entendiam os princípios básicos de comunicação via internet. O HTTP havia sido projetado para atender todas as necessidades que o CORBA, SOAP e outros procuravam suprir. O REST, por se basear em comunicação voltada a HTTP, pode passar por firewalls e é facilmente integrado com outros sistemas independente da plataforma. Deixando estes "conflitos" de lado, a ideia geral do REST é que ele seja um espelho do design da aplicação, onde cada recurso possui uma URL única e de fácil compreensão. Aqui o termo "recurso" é o que equivaleria a um método em uma arquitetura RPC.

Esses são os conceitos. Agora, o que tudo isso significa na prática? Muito simples, principalmente para quem já conhece Delphi. A maioria dos desenvolvedores está acostumado a desenvolver aplicações client/server com Firebird e Delphi Win32. Aí vai o BD, Data Module, talvez até um TMtsDataModule/TRemoteDataModule/TProviderDataModule, e uma interface que recebe dados do server, um class(TForm). Nossa interface não fica trafegando pela rede cada vez que um usuário clica em um botão, por exemplo, pois isso não existe em ambiente Desktop. Mas incrivelmente, se parar para pensar, é exatamente assim que a Web funcionava. Ao invés disso, as estações clientes rodam executáveis pré-instalados que apenas trocam dados com o servidor (o que chamamos de thin-clients). O DataSnap do Delphi 6 introduziu o suporta ao SOAP em IAppServer (veja meu artigo da edição 70), ou seja, dados poderiam ser trafegados em pacotes de dados XML (DataPackets) sobre o protocolo HTTP, para trafegar pela internet sem serem bloqueados por firewalls. Surgiram os Smart-clients no Delphi. Você pode criar uma aplicação Desktop que recebe dados de um servidor Web que pode estar localizado do outro lado do mundo, mas sua tela não fica trafegando pela rede junto com os dados, apenas os dados são trocados, em XML, usando SOAP. A vantagem do REST sobre o SOAP, nesse caso, é que o server não trabalha mais com métodos (apesar de IAppServer ser uma baita mão na roda), e sim, com recursos, descobertos intuitivamente. O cliente está no comando, ele diz o que quer. É mais poder para o cliente, ele está mais "rico"!

Voltando um pouco mais atrás, o Delphi 5 introduziu uma tecnologia que eu considerava ser uma das mais poderosas já criadas. O Internet Express permitia que aplicações Web clientes se comunicassem com servidores DataSnap (na época chamado de MIDAS – Multitier Distributed Application Services) para obter dados estruturados em formato XML. Esses dados então chegavam a um browser que apresentava uma interface fixa HTML, já formatada, que apenas manipulava um pacote de dados (XML DataPacket) recebido do servidor, usando apenas JavaScript. Não havia postbacks ao servidor enquanto os dados eram manipulados nos cliente. Pois é, a ideia do Internet Express ressuscitou, com outro nome.

O ADO.NET Data Services (também chamado de Astoria) é a tecnologia do .NET Framework que, em conjunto com o ADO.NET Entity Framework (tecnologia de mapeamento objeto relacional e persistência do .NET, concorrente do NHibernate), permite trabalhar com princípios do REST. Usando uma nova versão do ASP.NET AJAX, a 4.0 (hoje ainda em CTP), aplicações cliente rodando no browser podem estabelecer uma troca de dados com o servidor sem ficar trafegando a interface toda a cada requisição. O ASP.NET AJAX 4.0 consegue isso usando bibliotecas JavaScript que manipulam os dados recebidos do servidor Astoria (alguma semelhança com o falecido Internet Express?). Além disso, o AJAX 4.0 suporta o conceito de client-templates, algo muito parecido com os conhecidos server-controls do ASP.NET. Porém, ao invés dos controles rodarem no servidor ("run at server") e lá serem renderizados para HTML, eles nada mais são que elementos embutidos na própria interface HTML que, através das bibliotecas clientes JavaScript do AJAX, e uma pitadinha de CSS, exibem e manipulam dados totalmente no cliente, sem requisições desnecessárias e postbacks extras. Então, o que o DataSnap 2010 tem a ver com a história e por que ele está no título deste QU? O DataSnap 2010 agora suporta REST / JSON (bingo), o que significa que ele pode fazer boa parte do que o ADO.NET Data Services faz, abrindo a arquitetura das aplicações multicamadas, para termos assim clientes escritos e rodando nas plataformas mais heterogêneas possíveis.

Com isso, temos um novo cenário se consolidando no mundo Delphi: aplicações RIA – Rich Internet Applications, onde os clientes não precisarão de muita coisa além de um browser atualizado e uma boa conexão à internet. Nada de instalação, configuração, visitas e altos custos para manutenção, atualizações off-line, service packs, updates, altos investimentos em hardware, software etc. Uma aplicação Web terá uma interface tão rica e atraente como é hoje, ou melhor dizendo, era ontem, uma aplicação Desktop. O cliente recebe uma tela, e através de chamadas assíncronas, troca dados com um servidor. Os sites de redes sociais são um exemplo claro dessa arquitetura. E tudo isso tem uma íntima relação com outro padrão que já está firmado – Cloud Computing. Isso fará com que as pessoas, desenvolvedores e empresas revejam seus conceitos sobre o que é um software. Ou descendo um pouco, o que é um sistema operacional. Por que usar um SO se tudo o que eu preciso é um browser atualizado e uma boa conexão internet, para poder usufruir de uma infraestrutura nas nuvens, onde tudo é processado e armazenado? Nada de altos custos com hardware, manutenção, infraestrutura, isso fica por conta de DataCenters.

Finalizando, não importa quais tecnologias você usará em um futuro próximo, se DataSnap 2010 com Delphi 2010 ou ADO.NET Data Services com Delphi Prism / C# no VS 2010, todos vão exercer o mesmo papel no servidor. E no cliente, não importa se vai usar AJAX, Silverlight, VCL Win32, Mobile. O que importa é que a arquitetura já está definida, e muito bem definida. E o melhor, tudo isso se baseia em padrões abertos do mercado, já consolidados há anos, como HTTP, XML e JavaScript. Não existe como escapar disto, mas isso, companheiro desenvolvedor, é um ponto positivo. Literalmente, RIA. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Delphi 2010

Novas units, Gestures e Multi-Touch

Resumo do DevMan

Neste artigo vamos acompanhar as novidades do Delphi em relação às novidades apresentadas no Windows 7. Com um destaque especial ao novo recurso de Gestures. Agora é possível, em Delphi, criar aplicações Touch, Multi Touch e até mesmo definir Gestures, ou gestos, que se transformarão em ações. Mas o Delphi 2010 traz mais do que isso. As APIs de acesso e manipulação de diretórios foram organizadas em estruturas do tipo Record, que centralizam e agilizam o acesso às rotinas mais comuns como listar arquivos, verificar se um caminho é válido e muito mais.

Rodrigo Carreiro Mourão

rodrigocarreiro@rmfactory.com.br

Apaixonado por tecnologia da informação. Atualmente é sócio diretor da RM Factory Soluções em TI, empresa especializada em consultoria na área de TI e desenvolvimento de softwares customizados, parceira da TDS Tecnologia - RJ. Possui os títulos de BDS2006 Win32 Product Certified e Borland Instructor Certified. Editor Técnico da Revista Clube Delphi e Coordenador da área de Delphi no sistema de suporte on-line da DevMedia. Palestrante da Borland Conference 2007. Mantenedor da comunidade www.delphisophp.com, a primeira do Brasil 100% feita com Delphi for PHP. Autor de diversos artigos técnicos veiculados nos principais sites e revistas sobre Delphi do Brasil .

Seguindo na linha de apresentação das novidades do Delphi 2010 vamos falar um pouco sobre as novidades na *VCL* e *RTL*. Foram adicionadas novas *units* na *VCL*. Algumas destas *units*, como a *Direct2D* e *D2D1*, foram adicionadas para que possamos utilizar novos recursos do *Windows 7*, neste caso a nova *API* gráfica *Direct2D*. Um artigo seria pouco para estudarmos todas as novas *units* a fundo, assim eu mostrarei aqui apenas um rápido *overview* sobre elas destacando as principais classes e métodos presentes nestas *units*.

Do que trata o artigo

A evolução da interação do usuário com um computador não para. A nova versão do sistema operacional da Microsoft, o Windows 7, traz recursos que oferecem aos sistemas nele executados, meios de utilizar essas inovações e o Delphi se atualizou nesta versão 2010 para tirar proveito dessas opções. Mas não é só isso, temos melhorias simples, mas importantes na área de actions e manipulação de arquivos e diretórios.

Para que serve

Após o lançamento do IPhone, a moda é o TouchScreen. Ele pode ser utilizado para facilitar a usabilidade de um sistema, oferecendo meios que vão além do clicar e arrastar, tornando seu uso mais agradável e pessoal. No Delphi agora temos componentes que permitem tratar os toques e gestos e transformá-los em ações dentro de nossos sistemas.

Em que situação o tema é útil

No desenvolvimento de sistemas onde a premissa seja facilitar a usabilidade sem exigir intimidade com mouse ou teclado. Um exemplo disso são máquinas de quiosques ou terminais bancários, onde um usuário qualquer pode simplesmente tocar a tela, ou arrastar seu dedo sobre a mesma para obter alguma informação.

IOUtils.pas

Como sempre, a intenção da equipe de desenvolvimento do Delphi é trabalhar para facilitar a vida dos desenvolvedores. Essa *unit* possui classes que expressam bem esta intenção. Trabalhar com arquivos e diretórios ficou ainda mais fácil, principalmente para quem trabalha com projetos orientados a objetos. Nela podemos destacar três novos tipos, são eles: *TDirectory*, *TPath* e *TFile*. Os nomes são bem intuitivos. O *record TDirectory* possui métodos para criação, alteração e manipulação de diretórios. O *record TPath* possui métodos para trabalharmos de maneira simples com caminhos de arquivos e diretórios. Já o *record TFile* possui métodos para gerenciarmos arquivos. Eles são muito similares às classes presentes no namespace *System.IO* presente no .NET.

Na **Listagem 1** vemos um exemplo de como podemos verificar a existência de uma diretório com o novo *record*. Para executar o exemplo crie um nova aplicação no Delphi 2010 e adicione apenas um botão e um *ListBox*. No Evento *OnClick* do botão insira o código da **Listagem 1**.

Listagem 1. Utilizado o record TDirectory

```
var
Path : string;
begin
  if not TDirectory.Exists('C:\temp\minha pasta') then
  begin
   ShowMessage('O diretorio não existe !!!')
  end
  else
  begin
   ListBox1.Clear;
   for Path in TDirectory.GetFiles('C:\temp\minha pasta', *.txt)
   do
     Listbox1.Items.Add('Nome do Arquivo '+Path);
  end;
end;
```

Veja que na **Listagem 1** usamos o método *Exists* do *record TDirectory*. Em nenhum momento criamos quaisquer objetos, simplesmente invocamos o método direto do *record*. Isso porque a maioria dos métodos que temos nos três *records* citados são estáticos, ou seja podem ser invocados direto do *record*. Voltando ao código verificamos a existência de uma pasta qualquer em nosso sistema. Caso ela não exista exibimos a mensagem ao usuário e saímos da rotina. Caso ela exista então voltamos a consumir outro método do *record TDirectory*. Desta vez o método *GetFiles*.

Neste método nós informamos dois parâmetros: o caminho do diretório que queremos fazer a busca e o filtro com a extensão dos arquivos que queremos localizar. A facilidade fica por conta deste método *GetFiles* em retornar uma lista passível de ser percorrida pelo *for in*. Então nós simplesmente percorremos a lista com um *for in* e para cada arquivo localizado adicionamos seu nome no *ListBox*. Simples assim!

Direct2D.pas

Talvez você não saiba, mas uma nova API gráfica foi adicionada no *Windows* 7. Esta nova API é chamada de *Direct2D*, para desenvolvimento de gráficos 2D no *Windows* 7. Seu objetivo é substituir as antigas API's GDI e GDI+ que são mais lentas.

Hoje a GDI e a GDI+ são responsáveis pelos efeitos 2D do *Windows*. A GDI + trouxe melhorias nas transparências e isso pode ser visto na interface do *Windows XP*. Porém continuamos a ter alguns problemas como falta de recursos mais avançados e desempenho uma vez que a *GDI* não é acelerada pela *GPU*.

A *unit Direct2D*.pas traz para nós uma nova classe, a *TDirect2DCanvas*. Ela é bem parecida com a já famosa *TCanvas*, para que possamos manipular e gerar interface gráficas mais eficientes e rápidas. Os controles da *VCL* não possuem o *Direct2D* habilitados por padrão. Eles estão usando o *GDI* baseado no Objeto *TCanvas*.

Para utilizar a nova tecnologia, você deve explicitamente adicionar suporte para *TDirect2DCanvas* a seus controles. O código da **Listagem 2** mostra como pintar uma figura gráfica em um *form* utilizando a classe *TDirect2DCanvas*. Para isso crie uma nova aplicação no Delphi 2010. Localize no *form* o evento *OnPaint*. Nele adicione o código da **Listagem 2**.

Repare que o código é bem similar ao que seria criado usando a antiga classe *TCanvas*, com exceção dos métodos *BeginDraw* e *EndDraw* que são chamados no início e fim da "construção" da interface respectivamente. Como dito anteriormente, o *Direct2D* foi adicionado no *Windows* 7, portanto quando for construir componente utilizando o *Direct2D* é

Listagem 2. Utilizado a Classe TDirect2DCanvas

```
var
   NovoCanvas : TDirect2DCanvas;
begin
   NovoCanvas := TDirect2DCanvas.Create(Canvas, ClientRect);
   NovoCanvas .BeginDraw;
  try
   NovoCanvas .Brush.Color := clBlue;
   NovoCanvas .Pen.Color := clBlack;
   NovoCanvas .Rectangle(150, 150, 250, 250);

  finally
   NovoCanvas .EndDraw;
   NovoCanvas .Free;
  end;
end;
```

Nota do DevMan

O GDI, ou Graphics Device Interface, é um dos três subsistemas principais do Microsoft Windows. É um padrão desse sistema operacional para representar objetos gráficos e transmiti-los para dispositivos de saída, como monitores e impressoras. É responsável por tarefas como desenhar linhas, curvas e fontes, providenciando todo um API específico para a execução dessas operações. No Delphi a classe TCanvas é responsável por consumir esses recursos da API

Nota do DevMan

GDI+ - Sucessora da GDI, foi introduzida no Windows XP e trouxe grandes melhorias, como antialiasing de gráficos 2D, criação de gradientes e suporte muito melhor aos formatos de imagem como o JPEG e o PNG (inclusive foi na GDI+ que ocorreu a famosa falha que poderia fazer com que a exibição de uma simples imagem JPEG na internet permitisse a instalação de um malware no sistema afetado).

interessante testar se o ambiente que a aplicação está rodando suporta o *Direct2D*. Isso pode ser feito utilizando o método estático *Supported*. Observe na **Listagem 3** um exemplo simples desta verificação:

Listagem 3. Verificando o suporte a Direct2D

```
if TDirect2DCanvas.Supported then
    ShowMessage('Direct2D Suportado')
else
    ShowMessage('Direct2D Não Suportado');
```

Dica

Ainda nesta área de interação gráfica temos interessantes novas units: Direct3D.pas, D3DX9.pas, D2D1.pas.

Gestures.pas

Umas das grandes novidades do *Windows 7* foi a adição do suporte a *multi touch*. A maneira de interagirmos com os dispositivos está evoluindo e as aplicações para estes dispositivos também. Seguindo esta filosofia e acompanhando a evolução da tecnologia, estão presente no Delphi 2010 três *unit*s que possuem classes com métodos e propriedades específicas para trabalhar com esta nova filosofia de iteração.

Ainda neste artigo eu dedicarei uma seção exclusiva para falarmos de *Gesture* e *Touch*, mas acredito que seja válida uma rápida descrição das principais classes presentes nestas *unit*s. Vamos começar pela *unit Gestures.pas*. Nela temos a presença entre outras da classe *TGestureEngine*.

TGestureEngine é a classe base para todos os mecanismos que interagem com os "gestos" feitos pelo usuário na tela. Esses mecanismos descendentes de *TGestureEngine* é que fazem a ponte entre os dispositivos que são sensíveis ao toque e os controles da VCL.

Esta engine geralmente será conectada e um mecanismo coletor de gestos (veremos mais a frente) que armazenará os pontos que compõem o gesto para que enfim possamos associá-lo a uma ação. Sim eu concordo que tudo isso parece muito complicado, mas não se preocupe. Eu estou falando de como as coisas funcionam nos bastidores, para nós programadores tudo isso será transparente.

GestureMgr.pas

A *unit GestureMgr.pas* traz a principal classe, na verdade componente, para se trabalhar com gestos. O componente a que me refiro é o componente *GestureManager*. A função deste componente na verdade é ser, como o nome sugere, o administrador ou gerenciador dos gestos por nós criados na aplicação. A **Figura 1** mostra o componente *GestureManager* e seu diálogo para a criação de *CustomGestures*. Este diálogo pode ser chamado dando um duplo clique sobre o componente *GestureManager* ou clicando com o botão direito sobre ele e selecionando a opção *CustomGesture*. Ao clicar no botão create do diálogo da **Figura 1** o editor para criação de gestos será chamado. A **Figura 2** mostra o diálogo para criação de gestos personalizados.

Nota do DevMan

GPU (Graphics Processing unit, ou Unidade de Processamento Gráfico), conhecido também como VPU ou unidade de processamento visual, é o nome dado a um tipo microprocessador especializado em processar gráficos em computadores pessoais, estações de trabalho ou videogames. Uma GPU normalmente é utilizada em placas de vídeo, nas versões simplificadas são integradas diretamente em placas.

Mais uma vez gostaria de citar que não entraremos em detalhes sobre estes componentes, pois os veremos em uma seção exclusiva sobre isso adiante.

GestureCtrls.pas

Na unit *GestureCtrls.pas* estão as classes dos novos componentes usados para trabalharmos com gestos. A **Figura 3** mostra os novos componentes presentes na *unit GestureCtrls.pas*, são eles: *TGestureRecorder, TGestureListView e TGesturePreview*.

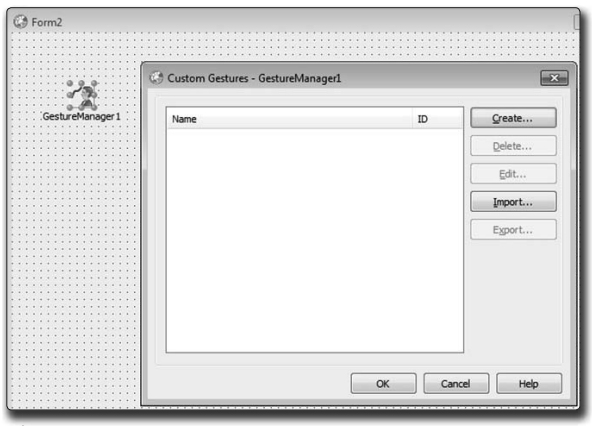


Figura 1. Componente GestureManager em ação

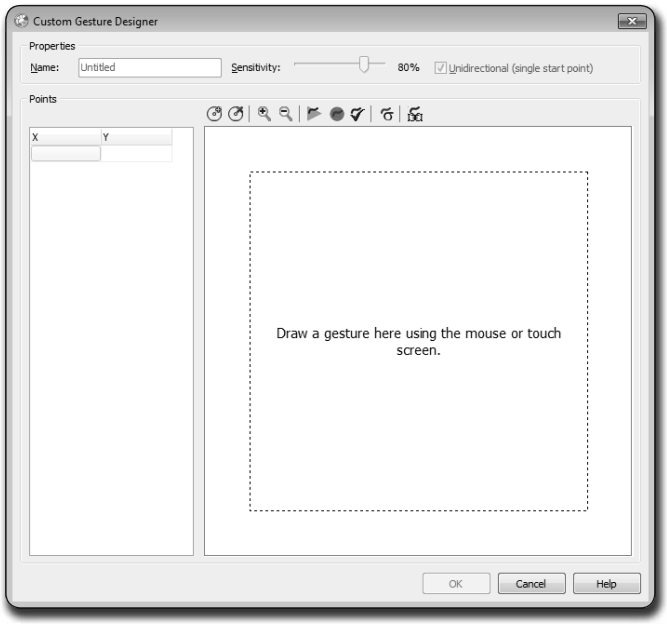


Figura 2. Editor para criação de gestos personalizados

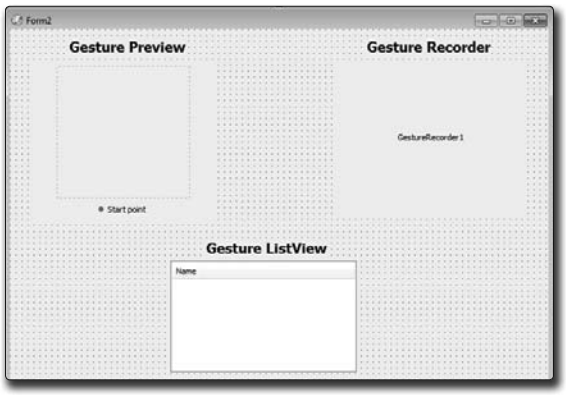


Figura 3. Novos componentes para trabalhar com Gesture

Nota

No tópico Natural Input deste mesmo artigo, veremos como trabalhar com estes componentes e veremos a diferença entre Touch, Multi-Touch e Gesture.

Keyboard.pas

Novamente seguindo a evolução de como interagimos com as máquinas e dispositivos, o Delphi 2010 traz mais uma nova *unit*, *Keyboard.pas*, que tem a classe *TTouckKeybord* que na verdade é um componente: o *TouchKeybord*.

O componente *TouchKeyboard* foi criado para facilitar o uso de telas sensíveis ao toque, substituindo o teclado físico presente na máquina do usuário. O *TouchKeyboard* fornece ao usuário um teclado virtual que pode ser exibido ou ocultado através do pressionamento de um botão por exemplo. Além disso, ele pode ser alterado para atender suas necessidades ou mesmo, caso você queira poderá estendê-lo e criar seu próprio teclado virtual.

O componente *TouchKeyboard* funciona como um teclado normal com exceção das teclas modificadoras como é o caso do *Shift* e *Fn*. Estas teclas são definidas como *toggable*, ou seja, você não precisa mantê-las pressionadas para usá-las. Assim, por exemplo, quando *Shift* é pressionada uma vez ela é ativada e a próxima tecla pressionada "levará isso em consideração". Uma letra pressionada, por exemplo, seria exibida como maiúsculas. Uma vez que o usuário toca em uma das teclas de letra, a tecla *Shift* será desativada e as teclas de letras retomam a sua escrita normal. As **Figuras 4 e 5** mostram as duas versões do componente *TouchKeybord*, o teclado padrão e o teclado numérico. Para fazer esta modificação basta alterar a propriedade *layout* para *Standard* ou *NumPad*.

TAction

Além de todas as novidades no IDE, novas *unit*s e novos componentes o Delphi 2010 trazem também muitas novidades nos componentes já existentes. Alguns possuem novas propriedades outros novos comportamentos e chegamos até ter novidades em propriedades, como é o caso da propriedade *Action*.

A partir desta nova versão as propriedades *Action* de todos os componentes agora apresentam um novo comportamento. Quando em nossa aplicação temos um *ActionList* ou um

Figura 4. Teclado Virtual versão Standard

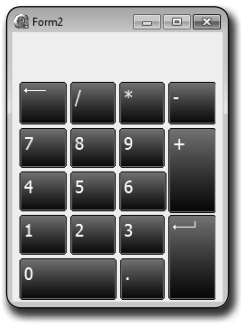


Figura 5. Teclado Virtual versão NumPad

ActionManager nós temos a possibilidade de criar uma nova ação direto da propriedade *Action* no componente.

Caso na sua aplicação haja mais de um *form* basta dar uses no *form* que possui o *ActionList* ou o *ActionManager*. Assim este recurso passa a estar disponível nas propriedades *Action* dos componentes deste novo *form*. A **Figura 6** mostra este novo recurso em ação.

Natural Input

Hoje enquanto você lê este artigo milhares de pessoas estão usando uma das grandes invenções do século, o *mouse*. Atualmente isso é apenas um coadjuvante, as novas gerações como a minha não pegaram a época de *softwares* em modo caractere. Nós já fomos apresentados aos ambientes gráficos. Se voltarmos mais ainda no tempo, chegaremos até os cartões perfurados. Quem diria que hoje estaríamos

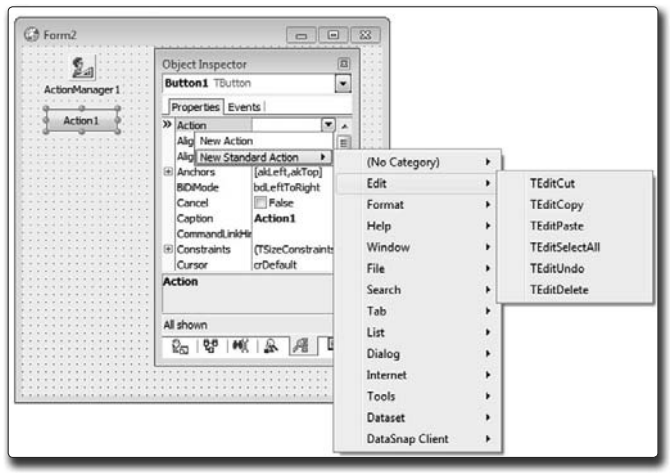


Figura 6. Nova feature nas propriedades Action dos componentes

falando sobre uma nova forma de interação com as máquinas, os dedos.

A forma com que o ser humano interage com os computadores e dispositivos vem evoluindo com o passar do tempo. O novo sistema operacional da *Microsoft* é prova disso. O *Windows 7* vem com suporte a *Multi Touch*. Mas até agora falamos em *Touch*, *Multi Touch* e *Gesture*. Não seria isso tudo a mesma coisa? Não!

Vamos às diferenças:

Touch

No sistema operacional *Windows* a atenção, ou melhor, o foco só pode estar presente em um controle por vez. Isso era o comportamento padrão antes do Win7. Assim o ponteiro do *mouse* só poderia ser usado em um ponto específico da tela. Neste contexto um dispositivo *Touch Screen* é apenas sensível ao toque. O que quero dizer é que num dispositivo *Touch* os dedos, ou melhor, um dedo irá substituir o *mouse*.

Há com isso uma diferença nas aplicações desenvolvidas para este tipo de dispositivo. Geralmente possuem botões grandes e a aplicação é voltada para o estilo *wizard* (assistente), de forma que a iteração com o usuário seja facilitada. Este tipo de aplicação e muito comum em Pontos de Venda, Quiosques com informações turísticas etc. Aqui no Rio de Janeiro, por exemplo, temos em algumas estações do metrô um sistema sensível ao toque com informações turísticas.

Multi Touch

Esta sim é uma grande revolução, digo isso hoje enquanto eu escrevo este artigo, pois na seção links você poderá conferir o que nos aguarda num futuro não muito distante. Os dispositivos *Multi Touch* têm como diferencial a possibilidades de reconhecer e interpretar múltiplos toques simultâneos na tela do dispositivo. Isso permite que você possa interagir com a aplicação usando as duas mãos. O Filme *Minority Report* é um exemplo. Claro que lá a coisa já estava numa fase mais avançada onde não precisávamos tocar na tela diretamente.

Agora vale lembrar que o sistema operacional tem que ter suporte a essa tecnologia e o *Windows 7* já nos dá este suporte. Outro detalhe importante e que o hardware também deve possuir esta capacidade de reconhecer múltiplos toques.

Gesture

Por fim, *Gesture*. Como o nome diz, eles nada mais são do que gestos. Estes gestos são na verdade movimentos que serão feitos no dispositivo, no caso a tela sensível ao toque, e que dispararão um evento. Assim, utilizando gestos podemos programar a aplicação para, por exemplo, fechar o *form* caso o usuário faça um movimento de círculo na tela. Isso é apenas um exemplo simples.

Poderíamos navegar nos registros de uma tabela só com os movimentos dos dedos. Os *Gesture* não dependem de um sistema operacional específico e sim que a aplicação tenha mecanismos para reconhecê-los. E advinha? O Delphi 2010 tem!

Nota do DevMan

Você sabe como funciona uma tela sensível a toque?
Existem 3 sistemas diferentes:

Resistivo – Elas funcionam através de pressão na tela que é reconhecida e traduzida como um comando específico. Essa tecnologia é utilizada, por exemplo, em uma das telas do Nintendo DS e nas telas de alguns caixas eletrônicos de banco. O sistema de identificação da pressão é feito utilizando-se duas placas bem finas, separadas por um afastador. Uma delas é feita de um metal e a outra de vidro. Entre elas passa uma corrente elétrica leve. Ao se encontrar na tela, as duas placas encostam-se naquele ponto. A mudança no campo elétrico é percebida e as coordenadas são enviadas ao computador. Essa informação é traduzida por um programa que transforma o toque em algum comando na tela. Por causa de como essa tela é produzida, ela pode ser utilizada com qualquer objeto que faça pressão. Muitas pessoas, por exemplo, utilizam canetas específicas. Esse é o caso do DS e de PDAs sensíveis ao toque ou até mesmo de celulares como o Nokia 5800 ou o HTC Touch. Mas também é possível tocar na tela com o dedo ou qualquer outro objeto.

Capacitivo - A tecnologia de telas capacitivas é a tecnologia utilizada nos iPhones e iTouchs. Ela funciona a partir de uma camada carregada de eletricidade, conhecida como camada capacitiva, posicionada sobre o painel do monitor. Ao se tocar na tela, parte desses elétrons é transmitida para o seu dedo, como se fosse um pequeno choque, mas muito leve a ponto de não se perceber. O computador então entende essa pequena descarga de eletricidade naquele ponto e calcula as coordenadas, que então são traduzidas como um comando para a tela do computador.

Ondas Acústicas Superficiais - Talvez a melhor tecnologia, justamente por permitir passar 100% da luminosidade da tela. Diferentemente das outras tecnologias, ela não posiciona nenhuma placa metálica sobre o monitor. Ao invés disso, são posicionados dois transdutores de cada lado da tela, um receptor de um transmissor, tanto dos lados quanto na parte superior e inferior, fechando um quadrado.

O transdutor transmissor manda uma pequena onda que é percebida pelo transdutor receptor. Essas informações formam então uma grade de coordenadas. Quando algo encosta-se à tela, os receptores percebem uma interrupção no envio das informações e informam automaticamente a coordenada do toque, traduzindo isso como um comando na tela.

Fonte: Pablo de Assis

Criando exemplo com Gesture

Para ilustrar vamos criar uma pequena aplicação para interagir com os movimentos do usuário. Com o Delphi 2010 aberto, crie uma nova aplicação. Não entrarei em detalhes de bancos e conexão a dados, pois foge ao escopo do projeto. Com a aplicação faça conexão com o banco de dados de sua preferência e adicione um *DataSet* (*Table* ou *Query*) e conecte a uma tabela qualquer.

Só a título de informação eu utilizei o componente *ADOConnection* para conectar ao banco de dados *dbdemos.mdb* que vem junto com a instalação do Delphi na pasta *C:\Program Files (x86)\Common Files\CodeGear Shared\Data\dbdemos.gdb*. Feito isso adicione todos os *Field*s ao *DataSet* e arraste-os para o *form*ulário. A **Figura 7** mostra uma sugestão de tela.

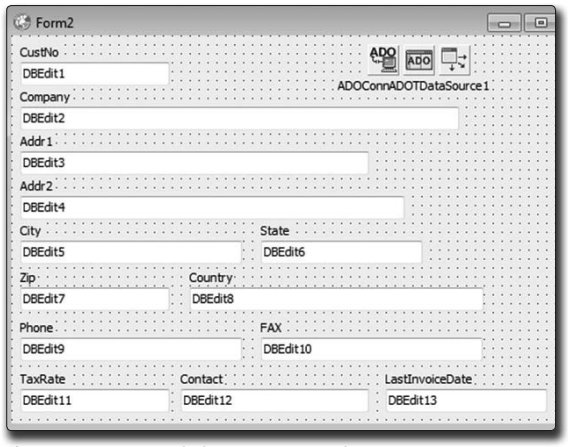


Figura 7. Sugestão de layout para a aplicação

Se você reparar bem não temos aí um *DBNavigator* ou qualquer outro botão para que possamos navegar entra os registros. Isso porque vamos utilizar gestos para isso. Então adicione um componente *ActionManager* e um componente *GestureManager*. Agora selecione o formulário e localize a propriedade *Touch* no *Object Inspector*. Clique no sinal de mais (+) para expandir a propriedade. Feito isso, aponte a propriedade *GestureManager* para o componente *GestureManager1*.

Ao executar este procedimento a propriedade *Gestures* passa a exibir uma coleção com os gestos padrões que vem pré-configurados na VCL. Clique nas reticências (...) ao lado da propriedade para invocar o diálogo com os gestos. A **Figura 8** mostra o diálogo em execução.

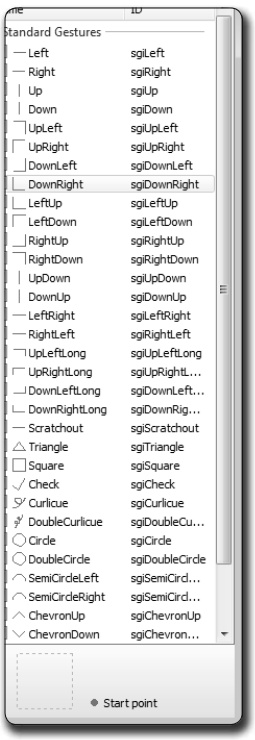


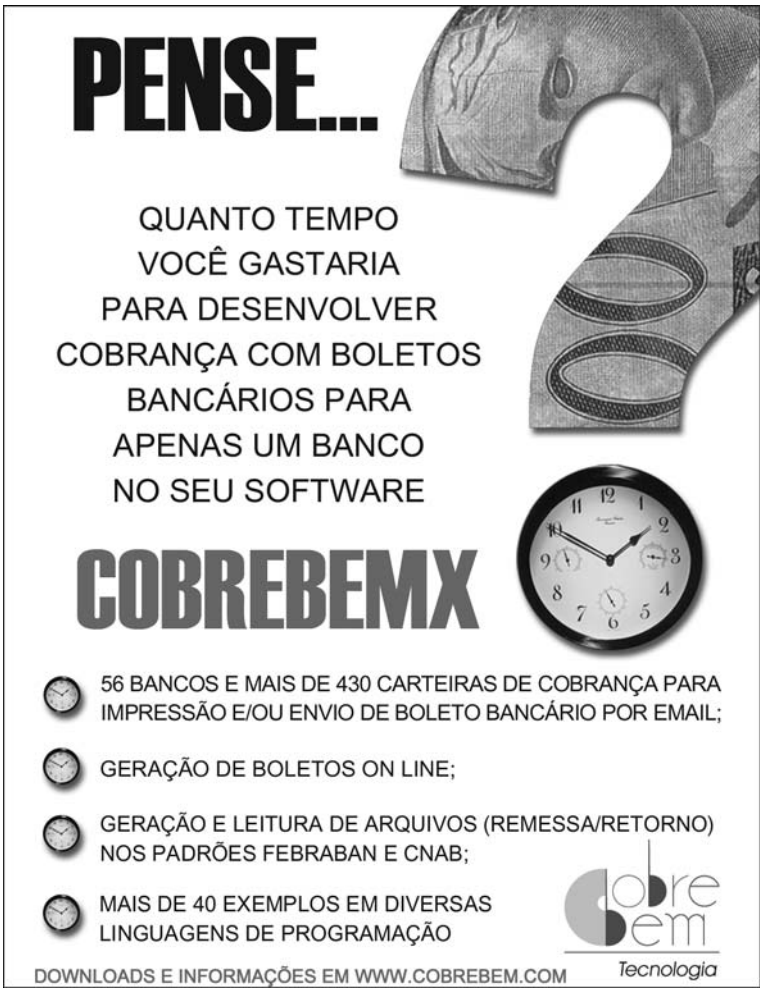
Figura 8. Gestos padrões da VCL

Neste exemplo usaremos os gestos *Up, Down, Left e Right* para respectivamente ir para o primeiro, último, anterior e próximo registro, respectivamente. A presença do *ActionManager* se deve ao fato de que precisaremos associar os gestos a uma *Action* que dispare a ação que queremos executar.

Como vimos podemos criar uma ação diretamente na propriedade *Action* dos componentes da *VCL*. Sendo assim, ainda com o *form* selecionado, clique no sinal de mais (+) ao lado da propriedade *Gesture*. Isso irá expandi-la exibindo a propriedade *Standard*. Expanda-a também para ter acesso aos gestos pré-definidos. Clique no combo do gesto *Left*, selecione *New Standard Action*, em seguida *DataSet* e por fim *TDataSetPrior* (a **Figura 9** mostra o menu expandido). Para o gesto *Right* faça o mesmo somente alterando o último passo para *TDataSetNext*. Associe o gesto *Up* com *TDataSetFirst* e por último associe o gesto *Down* com *TDataSetLast*.

Como nos não criamos ações e sim utilizamos as ações *Standard* do *ActionManager*, não teremos que digitar código algum, apenas associar a propriedade *DataSource* das ações para o nosso *DataSource1*. Para isso dê um duplo clique no *ActionManager*, lá estarão quatro ações que foram criadas no momento em que fizemos a associação com os gestos. Selecione as quatro ações e aponte a propriedade *DataSource* das ações para o componente *DataSource1*. A **Figura 10** mostra o procedimento.

Agora basta executar a aplicação e realizar os gestos no *form* para navegar entre os registros! Lembra que, para testes, os gestos podem ser feitos com o *mouse*.



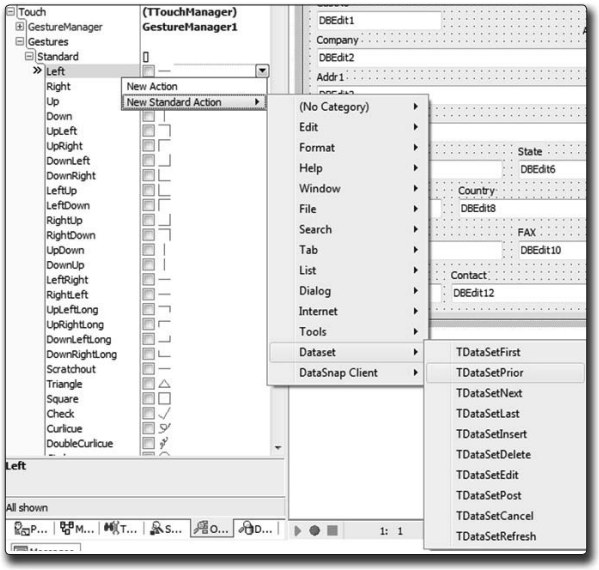


Figura 09. Associando os Gestos

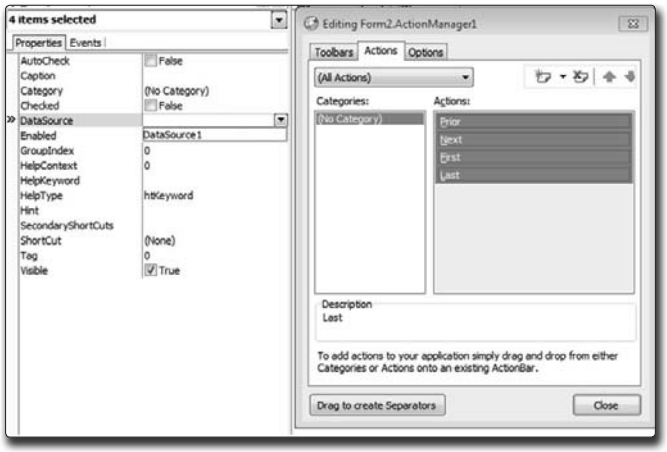


Figura 10. Associando as ações ao DataSource

Conclusão

Como foi visto, o Delphi se atualizou para as mais novas tecnologias de usabilidade de sistemas. Contudo isso não significa que você deve aplicar Gestures em todo e qualquer sistema que vir a desenvolver, principalmente para aplicações comerciais, onde a entrada de dados (digitação) deve ser rápida e prática. Para esses casos, nem mesmo mouse é indicado, pior ainda se você aplicar um Gesture e o monitor não oferecer suporte, o usuário terá movimentar o mouse na tela, de uma forma que ele não está habituado. Para a utilização desses recursos, o que vale é o bom senso, se coloque na posição de usuário um pouco e veja o que agrega ou não valor ao seu sistema.

Enfim, sem dúvida alguma a linguagem deu um enorme passo com a adição deste novo recurso. Muita coisa boa pode ser desenvolvida com o novos recursos. Na parte sobre Gestures e Touch temos a possibilidade de permitir que o próprio usuário final da aplicação possa criar seus gestos personalizados e muito mais, mas isso já renderia um novo artigo. Vale e muita apena dedicar parte do seu tempo de estudo para conhecer mais da ferramenta. Eu fico por aqui, um grande abraço e até o próximo artigo. ●

Links

Conheça o equipamento chamado de Sexto Sentido

http://www.youtube.com/watch?v=YVvJw4FJmUU

Projeto Natal da Microsoft, interação ao máximo

http://www.youtube.com/watch?v=QxC0mE0ML9E

Microsoft Surface

http://www.youtube.com/watch?v=Zxk_WywMTzc

Blog do autor

http://rodrigocarreiro.blogspot.com/ http://twitter.com/carreiromourao

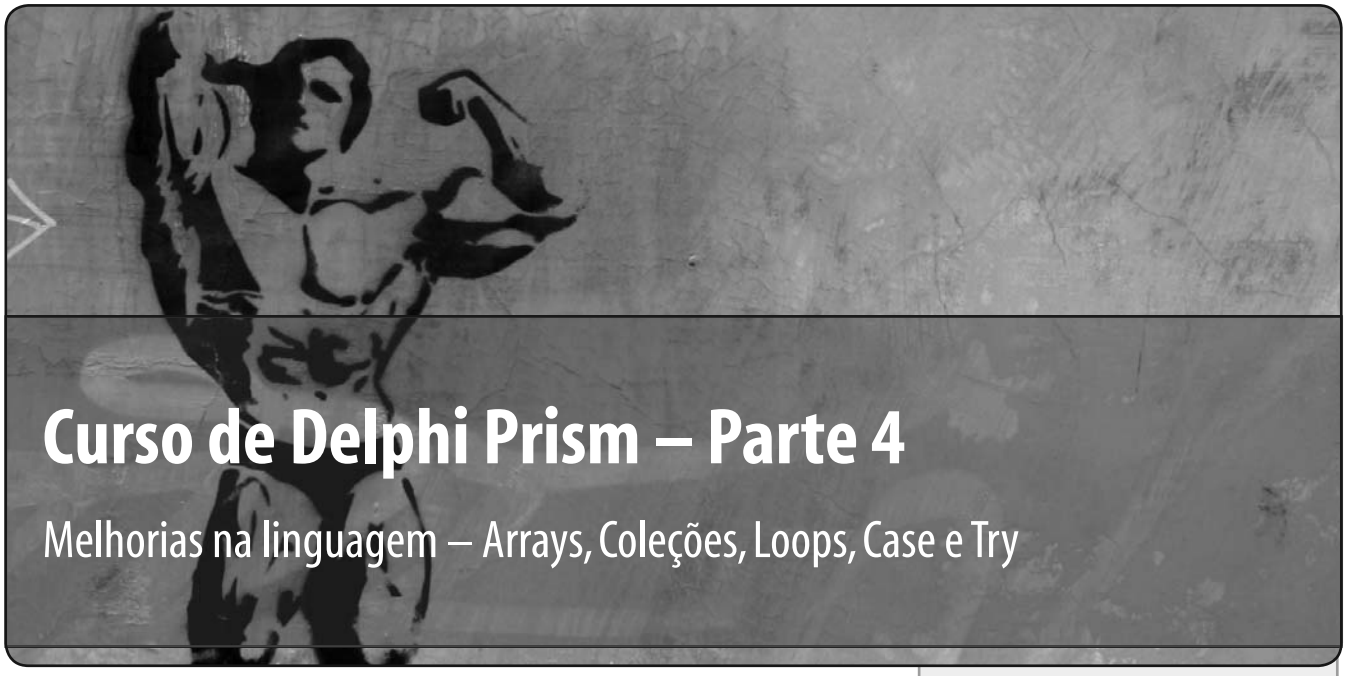
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET



Curso de Delphi Prism – Parte 4

Melhorias na linguagem – Arrays, Coleções, Loops, Case e Try

Guinther Pauli
guinther.pauli@gmail.com
Blog: http://guintherpauli.blogspot.com
http://twitter.com/guintherpauli
Tem 22 anos de experiência na área de desenvolvimento e arquitetura de software. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Kylix, certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD. Editor da Revista ClubeDelphi e da Revista .NET Magazine Brasil, professor de Pós-Graduação. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas. Foi arquiteto e um dos desenvolvedores, voluntário, do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Resumo do DevMan

Este artigo trata essencialmente nas melhorias (enhacements) da linguagem Delphi Prism (Oxygene). Veremos os incrementos feitos no for, try, arrays, coleções, case e também, é claro, algumas novidades.

Continuando nossa jornada pelas novidades do *Delphi Prism*, hoje vamos fazer alguns exemplos bem interessantes. É importante que você tenha lido as partes anteriores deste artigo, pois vou utilizar alguns recursos já apresentados, como tipos anônimos, *type-inference*, variáveis *in-line*, propriedades automáticas e construtores estendidos (o que chamamos no *C#* de *object initializers*) e mais. Além disso, vamos dar uma breve introdução ao uso de *Generics*.

Do que trata o artigo
Lançado pela Embarcadero, empresa que adquiriu a CodeGear, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio 2008. Nesta terceira parte do curso, apresento mais sobre os novos recursos da linguagem utilizada pelo Delphi Prism, que são inúmeros se compararmos com o Delphi for .NET tradicional ou mesmo Delphi Win32.

Para que serve
Comparar a sintaxe do antigo Pascal e Delphi tradicional com a moderna e poderosa linguagem suportada pelo Delphi Prism, conhecer os principais novos recursos da linguagem e vantagens.

Em que situação o tema é útil
O Delphi Prism abre uma porta incrível para os desenvolvedores Delphi, pois podem agora utilizar um IDE robusto, rápido e estável para construir aplicações para .NET, incluindo as novas tecnologias ASP.NET 3.5, ASP.NET AJAX, WPF, WCF, MONO etc. usando uma versão evoluída do Object Pascal, que mescla os melhores recursos do Delphi e do C# 3.0.

Um ponto importante a ser considerado para realizar os exemplos deste artigo. Como comentei, o foco aqui não é o tipo de *interface* gráfica ou projeto utilizado, o *Prism* suporta hoje *Console*, *Windows Forms*, *Web Forms*, *Web Services*, *Mobile*, *WPF*, e muito em breve *Silverlight* e o que mais a *Microsoft* inventar. O foco aqui é a linguagem. Como padrão, estou usando *ASP.NET Web Forms*. Então, sempre lembre de criar o projeto com *File>New Web Site*, escolher a linguagem *Oxygene* e um diretório e seguir adiante.

A metodologia que vou usar aqui é a seguinte: para cada recurso apresentado, vou criar um método. Quando for testar o código, apenas chame esse método no *Page_Load* do form. Quando for para o outro tópico, retire a chamada do método e substitua pelo novo. Além disso, muitos dos exemplos aqui apresentados vão usar alguns controles de tela para apresentar algum resultado, então, logo após criar seu *Web Site*, já coloque no *Web Form* os controles *Label*, *TextBox*, *ListBox* e *GridView*. Preparado? Mente aberta? *Delphi Prism* aberto? Então vamos em frente!

Arrays

Existem várias formas de implementar *arrays* no *Delphi Prism*. O tipo mais simples é o *array* sem limites (*unbound*), como visto na **Listagem 1.** Já a **Listagem 2**, mostra um *array* com limites (*bound*). No primeiro exemplo, os limites do *array* são definidos em *runtime*, já no segundo, o *array* é tipado e seus limites já são pré-definidos. Podemos ainda usar *tipos enumerados* para definir os limites do *array*. Na *VCL*, por exemplo, pessoalmente eu criava um *array* de *strings* para traduzir os diferentes tipos de estado de *TDataSetState* (aqui na **Listagem 3** vai um exemplo retirado dos fontes do projeto *DataCar* – veja seção *Links*). Basicamente, no *OnStateChange* do *DataSource*, verifico o estado do *TDataSet / TDataSource* e configuro uma mensagem na *StatusBar*. No *Delphi Prism*, fazemos algo parecido como na **Listagem 4**, aproveite para notar como criar um *tipo enumerado* no *Delphi Prism*. O interessante no *.NET* é que não usamos prefixos nos valores dos *tipos enumerados*, como ***ds****Edit*, ***ds****Insert* etc. Aqui o valor, quando referenciado, é precedido do seu tipo, como *TStatus.Edit*, ou outros exemplos do framwork (*Color.Red, CommandType.StoredProcedure*).

Uma outra excelente vantagem do *Delphi Prism* com relação ao *Win32* é a capacidade de usar *arrays in-line*. Basicamente podemos fazer tudo no próprio bloco de código de implementação. Veja alguns exemplos na **Listagem 5**, um código que adiciona no *ListBox* alguns números de uma série *Fibonacci*, onde o próximo número é sempre a soma dos dois anteriores, iniciando com 1 e 1.

Coleções no .NET

E é claro, você pode usar tipos muito mais interessantes para armazenar seus objetos em vetores/*arrays* do que usando as simples estruturas acima (*arrays* puros). No *.NET Framework*, temos muitas classes prontas para trabalhar com coleções. Vamos ver alguns exemplos. Primeiro, vamos declarar um novo tipo (**Listagem 6).** A seguir, vamos usar um *ArrayList* para adicionar *pessoas* a uma coleção e exibir essas pessoas na tela (**Listagem 7)**. O resultado pode ser visto na **Figura 1.**

Nota

Observe que usamos um for que vai do 0 ao 11, ou seja, precisamos conhecer e ter muito cuidado com os limites do array. Mais adiante vamos ver um recurso que permite varrer arrays e coleções de forma mais segura, sem extrapolar seus limites e sem conhecer o índice inferior e superior.

Listagem 1. Unbound arrays

```
method _Default.TesteBoundArray();
var
  // Unbound
  DuplaPadel: array of string;
begin
  // 2 elementos, índice 0 até 1
  // tamanho definido em runtime
  DuplaPadel := new string[2];
  DuplaPadel[0] := "Guinther";
  DuplaPadel[1] := "Rafael";
end;
```

Listagem 2. Bound arrays

```
type
  // tamanho fixo e definido com tipo
  TDuplaPadel = array[0..1] of string;
...
method _Default.TesteBoundArray();
var
  DuplaPadel: TDuplaPadel;
begin
  DuplaPadel := new TDuplaPadel();
  DuplaPadel[0] := "Guinther";
  DuplaPadel[1] := "Rafael";
end;
```

Listagem 3. Win32: Usando enums ao seu favor para definir limites do array

```
// definição do TDataSetState retirados dos fontes da VCL
TDataSetState = (dsInactive, dsBrowse, dsEdit, dsInsert,  dsSetKey,
  dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue,
   dsBlockRead,dsInternalCalc, dsOpening);
...
// fonte do projeto DataCar
procedure TFrmFichaPai.DtSrcStateChange(Sender: TObject);
const
 Estados : array [TDataSetState] of string = ('Fechado',
  'Consultando','Alterando','Inserindo','','','','','','','','');
begin
  btnIncluir.Enabled:=DtSrc.State in [dsBrowse,dsInactive];
  btnGravar.Enabled:=DtSrc.State in [dsInsert,dsEdit];
  btnCancelar.Enabled:=DtSrc.State in [dsInsert,dsEdit];
  btnExcluir.Enabled:=DtSrc.State in [dsBrowse];
  btnProcurar.Enabled:=DtSrc.State in [dsBrowse,dsInactive];
  StsBr.Panels[0].text:=Estados[DtSrc.state];
end;
```

Listagem 4. Bound arrays com enums no Delphi Prism

```
type
  TStatus = (Edit, Browse, Insert);
  TStatusMessages = array [TStatus] of string;
...
method _Default.TesteEnumBoundArray();
var
  Mensagens: TStatusMessages;
begin
  Mensagens[TStatus.Browse] := "Pesquisando registro";
  Mensagens[TStatus.Edit] := "Editando registro";
  Mensagens[TStatus.Insert] := "Inserindo registro";
end;
```

Listagem 5. Arrays In-line

```
method _Default.TesteInLineArrays();
var
  i: Integer;
begin
   var Fibonacci: array of integer := [1, 1, 2, 3, 5, 8, 13, 21,
   34, 55, 89, 144];
   for i := 0 to 11 do
   ListBox1.Items.Add(Fibonacci[i].ToString());
end;
```

Nota do DevMan

Se observar no código, foi comentado que você deve adicionar o namespace System.Collections ao uses para usar ArrayList. Porém, isso não é obrigatório, você pode simplesmente omitir a declaração no uses e dar o caminho completo da classe (com o namespace) na sua criação, como no exemplo:

```
var Pessoas := new System.Collections.ArrayList();
```

Listagem 6. Tipo, com propriedades automáticas

```
type
...
Pessoa = public class
  public
    property Nome: String;
    property Idade: Integer;
  end;
```

Listagem 7. Usando ArrayList

```
method _Default.TesteArrayList();
begin
    var Eu  := new Pessoa(Nome := "Guinther", Idade := 31);
    var Ben := new Pessoa(Nome := "Benjamin", Idade := 21);
    // uses System.Collections
    var Pessoas := new ArrayList();
    Pessoas.Add(Eu);
    Pessoas.Add(Ben);
    GridView1.DataSource := Pessoas;
    GridView1.DataBind();
end;
```

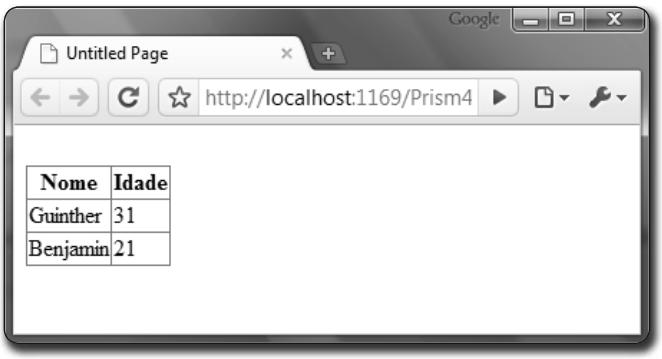


Figura 1. ArrayList sendo exibido no GridView

ArrayList aceita objetos. Uma técnica muito mais poderosa, que surgiu no *.NET Framework 2.0*, é a capacidade de usar *Generics*. Basicamente, em linhas gerais, podemos usar uma coleção genérica onde o tipo de dado que vai ser usado na coleção é definido em ***tempo de compilação***. Isso dá performance, evita *RTTI* (que no *.NET* se chama *Reflection*), evita *type-casts*, que são lentos, pois temos uma coleção **fortemente tipada**. É simples de usar, veja o mesmo exemplo da **Listagem 7** agora adaptado para usar uma lista genérica (**Listagem 8)**. O resultado obtido será idêntico ao da **Figura 1.** *Generics* é um assunto a parte, ele não se aplica apenas a coleções (veja edição 110, seção *Ask the Expert*).

Temos ainda algumas outras coleções, como o poderoso *HashTable*, que permite armazenar duplas de dados no formato *key/value* (chave/valor). Imagine isso como uma tabela de um banco de dados. Você tem a chave primária (1, 2, 21, 1928) e pode usar ela para acessar os demais dados dos registros, com

Listagem 8. Usando List<T> (Generics)

```
method _Default.TesteGenericList();
begin
    var Eu := new Pessoa(Nome := "Guinther", Idade := 31);
    var Ben := new Pessoa(Nome := "Benjamin", Idade := 21);
    // uses System.Collections.Generic
    var Pessoas := new List<Pessoa>;
    Pessoas.Add(Eu);
    Pessoas.Add(Ben);
    GridView1.DataSource := Pessoas;
    GridView1.DataBind();
end;
```

Nota do DevMan

Generics são agora suportados no Delphi Win32, no Delphi 2009 e Delphi 2010.

a vantagem que a chave pode ser qualquer objeto, inclusive strings. Um *HashTable* usa um algoritmo interno extremamente poderoso para permitir essa implementação em coleções no *.NET*. Vamos a um exemplo, observe na **Listagem 9** que criamos um *HashTable*. A chave é um objeto, ou seja, pode ser um número, string, qualquer coisa. O valor também é um objeto, pode ser uma string, um objeto de um tipo definido (é o nosso caso, *Pessoa*) ou até mesmo uma coleção. Dentro do *HashTable* colocamos algumas pessoas, depois, pedimos ao usuário para digitar a chave no *TextBox* (por exemplo "*gp*"). Ao pressionar *Enter*, será dado um *Postback* na página e o nome da pessoa correspondente à chave será mostrado no *Label*.

Listagem 9. HashTable

```
method _Default.TesteHashTable();
begin
    var Pessoas := new Hashtable();
    Pessoas.Add("gp",new Pessoa(Nome := "Guinther", Idade := 31));
    Pessoas.Add("bc",new Pessoa(Nome := "Ben", Idade := 21));
    var Key := TextBox1.Text;
    if Key <> String.Empty then
    begin
        var Alguem := Pessoas[Key] as Pessoa;
        Label1.Text := Alguem.Nome;
    end;
end;
```

For

As estruturas de repetição foram bastante incrementadas no *Delphi Prism*. Vamos examinar o nosso tradicional *for*, em uma aplicação *Delphi Win32* (**Listagem 10**). Observe como podemos fazer agora no *Delphi Prism*, usando variáveis *in-line* no próprio *for* (**Listagem 11**). Não precisamos declarar a variável do *loop* na seção *var*, podemos declará-la e colocá-la diretamente no local onde vai ser usada. Além disso, podemos usar o *step* (como no *MSX basic* ou *TK85*), onde instruímos o compilador a fazer iteração de 1 a 10, porém com incremento de 2, o que fará o *ListBox* ser preenchido somente com os números ímpares (**Listagem 12)**. E claro, *downto* pode ser usado normalmente no lugar do *to* para fazer a iteração reserva.

For em coleções

Não foi à toa que iniciei este artigo falando de *arrays* e coleções, pois agora vamos trabalhar com algo intimamente

Listagem 10. For no Delphi Win32

```
procedure TForm1.TesteFor();
var
  i: integer;
begin
  // Preenche um ListBox com números de 1 a 10
  for i := 1 to 10 do
    ListBox1.Items.Add(IntToStr(i));
end;
```

Listagem 11. For no Delphi Prism

```
method _Default.TesteFor();
begin
    // Preenche ListBox com os números de 1 a 10
    for i: Integer := 1 to 10 do
      ListBox1.Items.Add(i.ToString());
end;
```

Listagem 12. For no Delphi Prism, com step

```
method _Default.TesteFor();
begin
    // Preenche ListBox com os números de 1 a 10,
    // porém, pula os números pares, usando Step
    for i: Integer := 1 to 10 step 2 do
      ListBox1.Items.Add(i.ToString());
end;
```

Listagem 13. For no Delphi Win32, varrendo uma coleção

```
procedure TForm1.TesteForColecao();
var
  i: integer;
begin
  // Coloca o nome de todos os componentes do form em um ListBox
  for i := 0 to ComponentCount - 1 do
    ListBox1.Items.Add(Components[i].Name);
end;
```

relacionado à manipulação delas. É muito comum precisarmos varrer coleções de objetos. Normalmente, quando varremos uma coleção, fazemos algo como na **Listagem 13,** em *Delphi Win32* (quem nunca fez o famosíssimo *for i:= 0 to Algo count – 1* que atire a primeira pedra – lembra do *ControlCount -1, ComponentCount -1, FieldCount -1?*).

Ou seja, você precisa conhecer os limites da coleção para poder fazer a varredura. Nesse caso, abre-se uma brecha para dois problemas: ou você esquece algum item da coleção, iniciando o *loop* não pelo primeiro elemento, ou ainda pode acessar um elemento fora da coleção, o que irá causar um *List Index out of Bounds* (veja **Listagem 14** e resultado na **Figura 2**).

Figura 2. É perigoso varrer um vetor/coleção se você não sabe o seu tamanho

Listagem 14. For no Delphi Win32, varrendo uma coleção, com problemas

```
procedure TForm1.TesteForColecao();
var
  i: integer;
begin
  // Coloca o nome de todos os componentes do form no ListBox
  for i := 1 to ComponentCount do // esqueci o - 1 e comecei em 1
    ListBox1.Items.Add(Components[i].Name);
end;
```

Nota do DevMan

For com Variants

Existe uma técnica no Delphi Win32 que talvez poucos conheçam, mas que ajuda a resolver esse problema. A ideia ganhou forças no Delphi 6, quando surgiu a unit Variants e separou todo o código que estava preso ao COM. A **Listagem 15** mostra como criar dinamicamente um vetor de strings em runtime, sem declarar o seu tamanho, que é configurado em runtime. A seguir, setamos o seu tamanho com VarArrayCreate e configuramos os valores de cada item. E finalmente, um for usa também funções especiais da unit variants para percorres todos os itens do array, sem se preocupar com seu tamanho, usando as funções VarArrayLowBound e Var ArrayHighBound para obter o índice inferior e superior do array, respectivamente. Essa técnica é muito utilizada em automação OLE, objetos COM e programação multicamadas com DataSnap.

Nota do DevMan

Arrays dinâmicos

O Delphi 4 introduziu mais um recurso relacionado aos arrays, os arrays dinâmicos. Basicamente declaramos o vetor/matriz sem especificar o tamanho, e em runtime vamos aumentando seu tamanho, como no exemplo:

```
var
  Nomes: array of string;
begin
  SetLength(Nomes,10);
```

Existe também uma outra técnica disponível no Win32, chamada open arrays (arrays abertos), muito usada para passagem de arrays para parâmetros de um método, sem especificar o tamanho. Usamos então as palavras-reservadas low e high para varrer o array aberto:

```
procedure ListaInteiros(const MeuArray: array of Integer)
var
  i: Integer;
begin
  for i := Low(MeuArray) to High(MeuArray) do
    WriteLn(MeuArray [i]);
end;
```

Listagem 15. Loops com Variants

```
// Uses Variants
procedure TForm1.TesteForColecaoVariants();
var
  i: integer;
  MITBlackJackTeam: variant;
begin
   MITBlackJackTeam := VarArrayCreate([1,4],varVariant);
   MITBlackJackTeam[1] := 'Guinther';
   MITBlackJackTeam[2] := 'Ben Campbell';
   MITBlackJackTeam[3] := 'Micky Rosa';
   MITBlackJackTeam[4] := 'Jill Taylor';
  for i := VarArrayLowBound(MITBlackJackTeam,1) to VarArrayHigh
   Bound(MITBlackJackTeam,1) do
     ListBox1.Items.Add(MITBlackJackTeam[i]);
end;
```

For Each

Um recurso muito bem-vindo, que já existia no *Delphi for .NET* é a possibilidade de usar o *for each*, idêntico ao *C#*. No *Delphi for .NET*, esse recurso se chamava *for in*. Eu realmente

não sei o porquê deste recurso no *Delphi* não ter sido chamado logo de cara de *for each*, como na maioria das linguagens *OO*. Pois bem, o pessoal do *Prism* não teve dúvidas e adotou o padrão. A ideia é simples, varremos uma coleção, seja lá de que tipo de objetos for, sem nos preocuparmos com os limites da coleção. Vamos a um exemplo na **Listagem 16.** Aqui eu estou criando um *DataTable* ("*Esportes*") do *ADO.NET* (algo como o *ClientDataSet* da *VCL*), totalmente no "braço", via código, em tempo de execução. Adiciono uma coluna, chamada "*Nome*" e a seguir adiciono dois registros à coleção *Rows* do *DataTable*, que representam os registros. Depois, usamos um *for each* para percorrer a coleção de registros, buscando pelos esportes cadastrados e adicionando os nomes no *ListBox*. Veja que para isso não precisamos conhecer os limites da coleção. A propósito, comentado no código, bem abaixo, está a implementação antiga, sem *for each*. Na **Listagem 17** eu coloquei um outro exemplo, agora usando *arrays*, é o mesmo exemplo da **Listagem 5** só que agora usando *for each*.

Nota

O exemplo da **Listagem 16** é didático é tem por objetivo mostrar o *for each* em ação. Em uma situação real, se o objetivo for simplesmente exibir os dados de um *DataTable* em um *ListBox*, você pode fazer o *DataBind* direto com a propriedade *DataSource* (que por nos bastidores, na verdade, vai usar um *for each*).

Listagem 16. For each

```
method _Default.TesteForEach1();
begin
  // uses System.Data
  var dt := new DataTable("Esportes");
  dt.Columns.Add(new DataColumn("Nome"));
  var r1 := dt.NewRow();
  r1["Nome"] := "Padel";
  dt.Rows.Add(r1);
  var r2 := dt.NewRow();
  r2["Nome"] := "BlackJack";
  dt.Rows.Add(r2);
  for each row: DataRow in dt.rows do
    ListBox1.Items.Add(row["Nome"].ToString());
  //sem for each
  //for i: Integer := 0 to dt.Rows.Count - 1 do
  //ListBox1.Items.Add(dt.Rows[i]["Nome"].ToString());
end;
```

Listagem 17. For each com arrays

```
method _Default.TesteForEach2();
begin
  var Fibonacci: array of integer := [1, 1, 2, 3, 5, 8, 13, 21,
   34, 55, 89, 144];
   for each num: Integer in Fibonacci do
     ListBox1.Items.Add(num.ToString());
end;
```

Case

Não preciso explicar o que é um *case*, esse vem desde o *Turbo Pascal*. O interessante é que podemos agora usar o *case* para *strings* (finalmente!), lembrando que no *Object Pascal* somente são aceitos tipos ordinais, como inteiros, *chars* etc. Você pode agora fazer algo como na **Listagem 18**. No *C#*, o *case* se chama *switch*, o que não é nada interessante comparado ao *Pascal*, é que no *C#* você precisa dar um *break* após cada teste do *case* (apesar de no C# *strings* também serem aceitas).

Vamos examinar outra nova *feature* no *case*. Você pode testar *tipos* em um *case*. Ao invés de ficar testando o tipo de um objeto

Nota do DevMan

Lembrando que o objeto que você for varrer deve implementar a interface IEnumerable, isso inclui praticamente qualquer coleção do .NET Framework. Além disso, você pode criar suas próprias classes e coleções que implementem IEnumerable e usar em um for each.

Listagem 18. Case agora pode ser usado com strings

```
method _Default.TesteCase1();
begin
    // Digite o nome de um filme no TextBox, pressione Enter
    // e veja um comentário
    var Filme := TextBox1.Text;
    case Filme of
        "The Island": Label1.Text := "A MS apresentou o Surface
          neste filme.";
        "21": Label1.Text := "Conseguiu desvendar o desafio dos
          quadros?";
        "The prestige": Label1.Text := "O lance de cortar o dedo
          foi sinistro.";
    else
        response.Write("Filme não disponível.");
    end; // case
end;
```

com inúmeros *if [objeto] is [tipo] then ... else if ...*, podemos usar um *case [objeto] type of*. Na **Listagem 19** preparei um exemplo bem interessante (caso tenha achado o exemplo dos filmes trivial), que vai servir não só para aprendermos esse tipo de *case*, mas alguns outros recursos do *.NET Framework*. Primeiro, veja como se obtém um número randômico no *.NET*, usando o *Random*. Neste caso, estou obtendo um número qualquer baseado no total de controles que estão na página (no momento são: *Label, TextBox, GridView* e *ListBox*). Com o *Random* sorteamos então um controle e aplicamos alguma operação nele. Se for um *TextBox*, mostramos nele mesmo qual a sua classe base usando o recurso de *Reflection*, com *GetType*, que retorna um *System.Type*. Se o objeto sorteado for o *ListBox*, *on-the-fly* criamos um *array* de *ListItens* (usando o recurso de *arrays inline* previamente explicado) para adicionar duas *strings* ao próprio *ListBox*. Se for o *GridView*, exibimos no próprio controle a relação de unidades lógicas da máquina, através da classe *Environment*. E finalmente, o *Label* exibe apenas um texto.

Nota

Existe uma outra construção muito interessante no Delphi Prism, também oriunda do C#, chamada using, que serve para proteger a alocação de recursos e encapsular um try finally.

Listagem 19. Case testando tipos

```
method _Default.TesteCase2();
begin
    var rd := new Random();
    var i := rd.Next(Page.Form.Controls.Count - 1);
    var Objeto := Page.Form.Controls[i];
    case Objeto type of
        TextBox : TextBox1.Text := TextBox1.GetType().BaseType.
          ToString();
        ListBox :
        begin
            var Itens: array of ListItem :=
                [new ListItem("Delphi"),new ListItem("C#")];
            ListBox1.Items.AddRange(Itens);
        end;
        GridView :
        begin
            GridView1.DataSource := System.Environment.
             GetLogicalDrives();
            GridView1.DataBind();
        end;
        Label : Label1.Text := "Hello";
    end;
end;
```

Try

Acredito que todos saibam usar o *try try except finally*, recurso usado para tratar exceções no seu código e garantir a liberação de recursos. A novidade é que você pode agora combinar tudo em um bloco só, diga adeus ao *try try*. Como no *C#*, onde podemos usar um *try catch finally*, podemos fazer *try except finally*. Veja um exemplo na **Listagem 20.** Aqui estou usando *ADO.NET* para conectar um banco de dados *SQL Server*. Instancio os componentes de acesso, abro a conexão e executo o comando *SQL*, tudo protegido. Se houver algum erro, ele será tratado, de acordo com o tipo. O *finally* vai garantir que a conexão seja fechada independentemente de erros acontecerem ou não. O objetivo aqui não é entender *ADO.NET*, apenas observe como podemos usar o *try except finally* de forma muito mais elegante.

Boa Prática

Tanto no Delphi Win32, Delphi Prism, C#, VB.NET ou qualquer outra linguagem orientada a objetos, sempre trate as exceções mais específicas primeiro, e por último a mais genérica, nesse caso, Exception. Se você tratar Exception primeiro, isso vai fazer com que qualquer tipo de exceção seja tratada na frente e você não vai ter chance de realizar um tratamento mais específico de acordo com o tipo levantado, como por exemplo, armazenar em log uma possível falha na conexão com o servidor de banco de dados.

Listagem 20. Try Except Finally, tudo junto

```
method _Default.TesteTry();
begin
    // uses System.Data.SqlClient
    var con := new SqlConnection({aqui vai a ConnectionString});
    var SQL := "delete Products where ProductID > 2";
    var cmd := new SqlCommand(SQL, con);
    try
        con.Open();
        cmd.ExecuteNonQuery();
    except
        on E: SqlException do
            Response.Write("Não foi possível encontrar o
             servidor ou login inválido:" + E.Message);
        on E: Exception do
            Response.Write("Erro ao executar comando SQL:"+E.
             Message);
    finally
        con.Close();
    end;
end;
```

Conclusão

Você já deve ter reparado que este curso não está abordando apenas o *Delphi Prism*, muitas coisas aqui apresentadas, mesmo no *Delphi Win32*, talvez não sejam conhecidas por muitos desenvolvedores. No entanto, vimos que no *Delphi Prism*, as técnicas são muito mais simples e elegantes de serem implementadas, sem gambiarras, com código mais simples, fluente, produtivo, flexível. Nada melhor do que aliar a rigidez do *Pascal*, a belíssima semântica do *Delphi* com o poder do *C#*, o que resulta em um belo produto chamado *Delphi Prism*. E esse é apenas o começo, vem muito mais por aí. Essa parte da série finaliza os novos conceitos da linguagem, lembrando que existem muitos outros. Aqui nos concentramos naqueles que provavelmente lhe causariam mais impacto, tanto no código quanto na maneira de pensar. A partir das próximas edições vamos partir para aplicações reais, com tecnologias novíssimas, prepare-se para usar *Delphi Prism* com *ASP.NET AJAX*, *ADO.NET Data Services*, *REST*, *JSON*, *Silverlight*, *WPF*, *ADO.NET Entity Framework*, *Azure*, *LINQ* e mais. O *Delphi Prism 2010* já está aí, o *VS2010* já está em *beta*, enfim, as coisas vão ficar bem interessantes por aqui, fique ligado no futuro, fique ligado na *ClubeDelphi*. ●

Links

Sistema DataCar 00
http://www.devmedia.com.br/articles/viewcomp.asp?comp=10472

Sistema DataCar Client/Server – Firebird e dbExpress
http://cc.embarcadero.com/Author/222668

Blog do autor
http://guintherpauli.blogspot.com
http://twitter.com/guintherpauli

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Programando com a API do Twitter

Integre suas aplicações Delphi com o microblog mais popular do mundo

Resumo do DevMan

O crescimento da comunicação virtual tem se expandido consideravelmente nos últimos anos, várias redes sociais têm surgido, porém, o Twitter vem se destacando como uma ótima fonte de informações, entretenimento e porque não dizer de negócios. Grandes empresas têm realizado constantes estudos para a utilização do Twitter como forma de se beneficiar, obtendo lucratividade, exibindo novidades e diferenciais de mercado. Pela web é possível encontrarmos centenas de programas e/ou utilitários que nos auxiliam na ação de "twittar". Neste artigo, veremos um exemplo de como interagir com sua API e criar nossa própria interface de interação.

Do que trata o artigo
Aprenda a utilizar a API do microblog Twitter, permitindo a criação de aplicações que possibilitem interação direta com o mesmo, ou facilitem o trabalho de postagem

Para que serve
Exibir e descrever de maneira simples e objetiva, maneiras de tirar proveito dos recursos da API do Twitter, permitindo que seja possível criar aplicações específicas para manipulação e gerenciamento do mesmo, ou adicionar recursos a uma determinada aplicação já existente

Em que situação o tema é útil
Desenvolvimento de utilitários, gadgets ou recursos extras de software. Interatividade e estudo sobre API's de desenvolvimento

Daniel Sobrinho Laporte
daniel.laporte@gmail.com/
daniel.sobrinho@devmedia.com.br
Editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em administração de bancos de dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005, desenvolvendo aplicações cliente/servidor e multicamadas em Delphi e Firebird.

Como sabemos, já há algum tempo um novo modelo de rede social vem ganhando enorme espaço na *web*. Antes tínhamos *Orkut* e *MySpace* por exemplo, porém a grande onda do momento agora é a rede social *Twitter*. Para quem ainda não conhece, o *Twitter* funciona como uma espécie de mini-blog, ou micro-blog, um diário eletrônico, onde a principal pergunta é: *What are you doing?* ou *O que você está fazendo?* Muitos apenas respondem a esta simples pergunta. Outros utilizam este meio para exibir informações, descrever

fatos ocorridos em seu dia-a-dia, novidades das mais variadas ou mesmo organizar movimentos, protestos entre tantas outras funcionalidades.

Para termos uma ideia, esta ferramenta tem tido um crescimento tão acelerado, que até os principais meios de comunicação do país têm aderido a ela. É possível encontrar na rede: jornais, atrizes, atores, músicos, humoristas, esportistas, desenvolvedores e muito mais. Com isto, temos grandes benefícios, pois nunca foi tão fácil interagir com pessoas que você nunca imaginava poder conversar, além de obter um grande volume de informações de forma extremamente rápida. O mais interessante de tudo é que estas informações são vistas pelas pessoas praticamente em tempo real.

Com todo esse crescimento e popularidade, a onda agora tem sido criar ferramentas, *gadgets*, entre muitas outras funcionalidades em conjunto com o *Twitter*. Os próprios navegadores, como o *Mozilla Firefox*, já possuem *plugins* como o *Twitterfox (https://addons.mozilla.org/pt-PT/firefox/addon/5081)*, por exemplo, que tornam a ação de "*twittar*" bem mais ágeis, permitindo a utilização por meio da barra de tarefas do próprio *Mozilla Firefox*.

Sendo assim, irei abordar neste artigo um pequeno exemplo prático e objetivo de como utilizar os recursos de *API* do *Twitter* em uma aplicação desenvolvida no *Delphi*, visto que atualmente há uma vasta fonte de exemplos práticos de utilização desta *API* integrada as mais variadas linguagens.

Conhecendo a API do Twitter

Após esta breve descrição, iremos visitar a página que disponibiliza toda a documentação necessária para podermos criar nosso exemplo. Para isso, acesse o endereço *http://apiwiki.twitter.com* **(Figura 1).**

Como podemos observar na **Figura 1,** possuímos três tópicos de *links* principais onde o primeiro, *Twitter API Documentation*, refere-se à documentação (será o que iremos utilizar no decorrer do artigo), o próximo *OAuth* abrange a parte de *FAQ* e

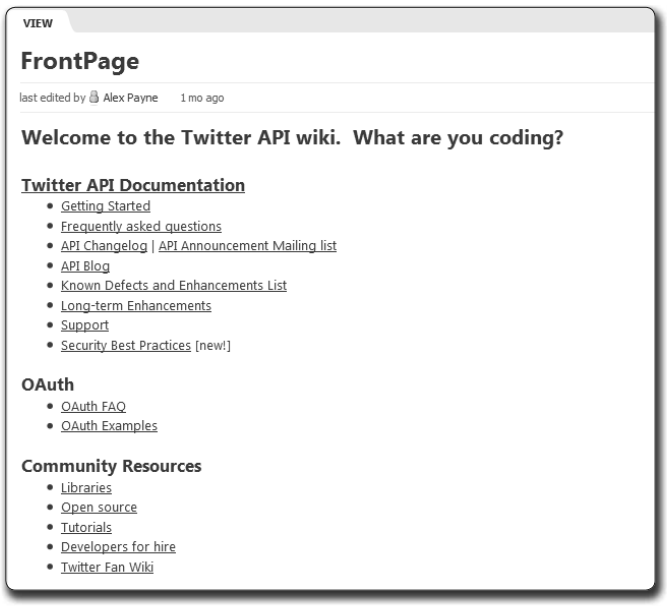


Figura 1. Página inicial da documentação da API

Nota

Nota: Mas o que isso tem a ver com o Delphi? Empresas usam o Twitter para divulgar sua marca, seus novos produtos, suas notícias, enfim, manter seus clientes conectados, a par de suas ações, atividades. Isso está sendo levado muito a sério, vemos nos meios de comunicação que a cada dia, empresas aderem ao Twitter. Como desenvolvedor Delphi, podemos integrar ao nosso produto (software) formas de interagir a empresa cliente com o Twitter, ou seja, nosso cliente com os clientes deles. E obviamente, nós com nossos clientes.

Nota do DevMan

Gadget (do Inglês: geringonça, dispositivo) é uma gíria tecnológica recente que se refere, genericamente, a um equipamento que tem um propósito e uma função específica, prática e útil no cotidiano, chama-se também de gadget algum pequeno software, pequeno módulo, ferramenta ou serviço que pode ser agregado a um ambiente maior.

exemplos, e por último a parte de contribuições da comunidade (*Community Resources*) onde tempos exemplos de bibliotecas, projetos *Open Source* e tutoriais. Como iremos utilizar a parte de estudos sobre a *API*, clique no *link Twitter API Documentation* para explorarmos um pouco das possíveis funcionalidades. Você será redirecionado para uma página como a da **Figura 2,** onde são listados os métodos disponíveis para utilização**.**

Selecione o *link users/show* dentro de *User Methods* para que possamos compreender um pouco da documentação. Vejamos na **Figura 3.**

Como podemos observar logo no topo da **Figura 3,** temos a descrição detalhada do que o método faz. Logo abaixo, é exibida a *URL* para a construção do método em questão. *Formats* descreve os tipos de formatos suportados, que no nosso caso

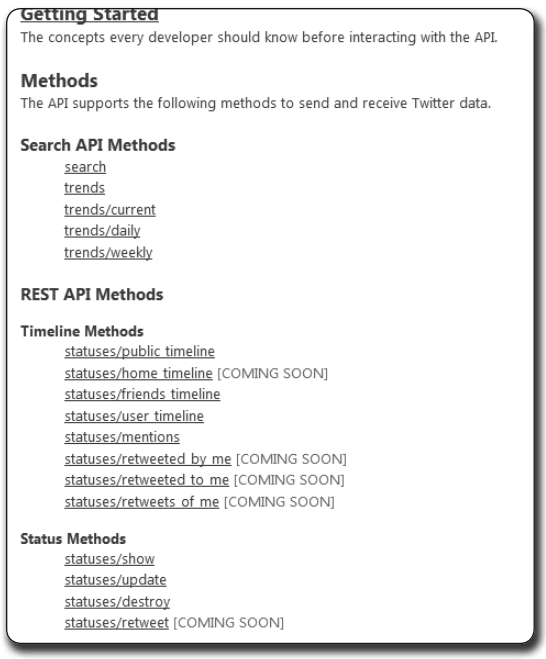


Figura 2. Lista de alguns métodos disponíveis na API do Twitter

abrange tanto *XML* quanto *Json.* Abaixo o tipo de método que pode alternar entre *GET* (recebe dados) ou *POST* (envia dados). O método em questão não exige autenticação, desta forma, é possível digitar a *URL* no *browser* e obter as informações do *XML* diretamente no seu navegador. *API rate limited* refere-se à taxa de limite de requisições, que no caso é de uma por vez. A parte que mais nos interessa neste exemplo é justamente *Parameters*, pois aqui estão contidas as instruções necessárias para o funcionamento da *URL*. Perceba que em nosso caso podemos optar por três parâmetros diferentes: *id*, *user_id* ou *screen_name*. Abaixo de cada parâmetro ainda possuímos um exemplo de utilização do *link* para obtenção das informações.

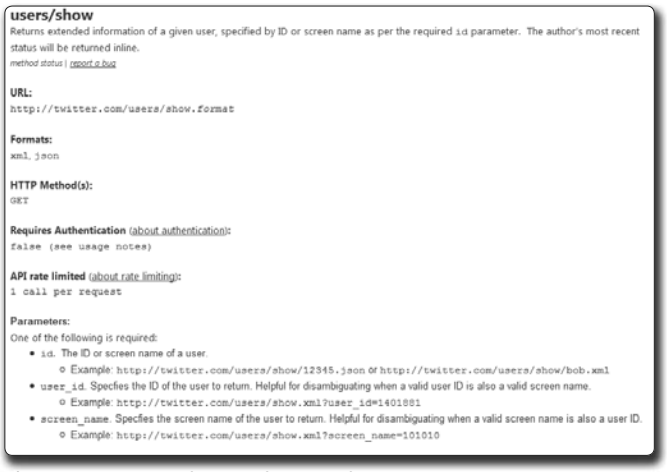


Figura 3. Descrição do método users/show

A **Listagem 1** exibe um exemplo de *XML* carregado no navegador, através do parâmetro *screen_name*, onde utilizei para o exemplo o meu próprio *screen_name.*

Repare que temos os *Nodes* ou nós no *XML*, na qual conseguimos obter informações das mais diversas como: nome, descrição, localidade, último *post* enviado, cores de texto e fundo entre muitas outras. Pois bem, iremos utilizar alguns destes métodos com determinados parâmetros para interagirmos com o *Twitter* através de nossa aplicação. Vamos ao exemplo.

Criando nosso exemplo prático

Abra então seu *Delphi* e crie uma nova aplicação VCL e renomeie o formulário principal para *frmPrincipal.* Altere seu *Caption* para *Utilizando a API do Twitter. A*ltere a propriedade *Position* para *poMainFormCenter* e *BorderStyle* para *bsDialog.* Salve seu projeto nomeando a *Unit* como *untPrincipal*.pas e o projeto como *APITwitter.*

Vamos dar início à adição de componentes no formulário. Insira um *Panel,* renomeie-o para *pnlTopo* e altere a propriedade *Align* para *alTop.* Dentro deste *panel* insira dois *Labels.* Altere a propriedade *caption* de um para *Nome de Usuário* e do outro para *Senha*. Feito isso adicione dois *Edits.* Renomeie-os *para edtUsuario e edtSenha* respectivamente.

Feito isso, adicione mais dois *BitBtns*, que podem ser renomeados para *btnLogin e btnLogout* e em seguida adicione um *GroupBox* e renomeio-o para *gbxPessoal*. Dentro deste *GroupBox* insira um *Memo* e chame-o de *memProfile*. Defina o *GroupBox* com *Align AlRight* e o *memProfile* como *AlClient.*

Para finalizar, adicione no formulário um *PageControl, renomeie-o para pclControle* e adicione neste *PageControl* três novas abas, a saber: *tbsMensagensDeAmigos*, *tbsMeusPosts* e *tbsMensagensDeAmigos*. Isso pode ser feito clicando com

Nota do DevMan

XML e JSON

XML é a abreviação da expressão eXtensible Markup Language. Esta é denominada como linguagem de marcação e trabalha baseada em TAGS. Mais informações sobre esta linguagem e padrões detalhados podem ser encontrados no site: http://www.w3.org/XML.

Json (JavaScript Object Notation) assim como XML, é um formato de intercâmbio de dados, que pode ser manipulado via JavaScript. Json é comumente utilizado por empresas que disponibilizam um grande volume de dados com grande performance. Empresas como Google e Yahoo tem adaptado o suporte a este formato. Embora seja frequentemente comparado à XML, é comum ver XML e Json trabalhando em conjunto em aplicações.

Listagem 1. Exemplo de arquivo XML obtido através do método users/show com parâmetro screen_name

```
<user>
<id>34479054</id>
<name>Daniel Laporte</name>
<screen_name>daniel_laporte</screen_name>
<location>Franca/SP</location>
<description>
Analista de sistemas e desenvolvedor de softwares. Apaixonado
por Deus, família e amigos. Obrigado por estar na minha lista de
pessoas que fazem a diferença...
</description>
<profile_image_url>
http://s3.amazonaws.com/twitter_production/profile_
images/258102704/IMAGEM-45_normal.JPG
</profile_image_url>
<url/>
<protected>false</protected>
<followers_count>19</followers_count>
<profile_background_color>1A1B1F</profile_background_color>
<profile_text_color>666666</profile_text_color>
<profile_link_color>2FC2EF</profile_link_color>
<profile_sidebar_fill_color>252429</profile_sidebar_fill_color>
<profile_sidebar_border_color>181A1E</profile_sidebar_border_
color>
<friends_count>25</friends_count>
<created_at>Thu Apr 23 01:25:07 +0000 2009</created_at>
<favourites_count>0</favourites_count>
<utc_offset>-14400</utc_offset>
<time_zone>Santiago</time_zone>
<profile_background_image_url>
http://static.twitter.com/images/themes/theme9/bg.gif
</profile_background_image_url>
<profile_background_tile>false</profile_background_tile>
<statuses_count>38</statuses_count>
<notifications>false</notifications>
<verified>false</verified>
<following>false</following>
<status>
<created_at>Thu Aug 06 11:14:33 +0000 2009</created_at>
<id>3164869723</id>
<text>
Test
</text>
<source>web</source>
<truncated>false</truncated>
<in_reply_to_status_id/>
<in_reply_to_user_id/>
<favorited>false</favorited>

<in_reply_to_screen_name/>
</status>
</user>
```

o botão direito sobre o *PageControl* e selecionando a opção *NewPage*. Selecione a primeira aba e adicione quatro *Labels* com os *Captions: Histórico, O que você está fazendo?, Caracteres restantes* e *140*. Feito isso adicione também dois *Memos (memPostar* e *memHistorico*) e por último um *BitBtn* (*btnAtualizarMeusPosts*).

Na segunda aba adicione um *BitBtn e* altere seu nome para *btnAtualizarPostsAmigos.* Adicione também um *Memo*, chame-o de *memFriendPost* e altere sua propriedade *Align para AlBottom.* Na terceira e última aba, insira um *BitBtn* (*btnAtualizarMsgAmigos*) e um *Memo* (*memDirectMsg*).

Finalizada a parte visual da nossa aplicação, vamos adicionar dois componentes de extrema importância no processo, são eles: um componente *IdHTTP,* que pode ser encontrado na paleta *IndyClients* e um componente *XMLDocument* que pode ser encontrado na paleta *Internet*. Após finalizar toda a parte de inserção dos componentes veja o resultado de nossa aplicação na **Figura 4.**

Nota do DevMan

O componente IDHTTP é descendente de TIdCustomHTTP que encapsula na verdade um cliente HTTP (Hypertext Transfer Protocol). Nele temos métodos e propriedades para realizar tarefas comuns de um cliente HTTP (Browser) como fazer uma requisição a uma url, realizar autenticação e muito mais.

Nota do DevMan

O componente XMLDocument é utilizado para representar um documento XML. Com ele é possível ler o conteúdo de um arquivo XML existente e mapeá-lo do início ao fim percorrendo cada uma dos seus nós e subnós. Ele ainda nos permite criar um arquivo XML novo, criar a estrutura de árvores com os nós e salvar o arquivo xml em disco.

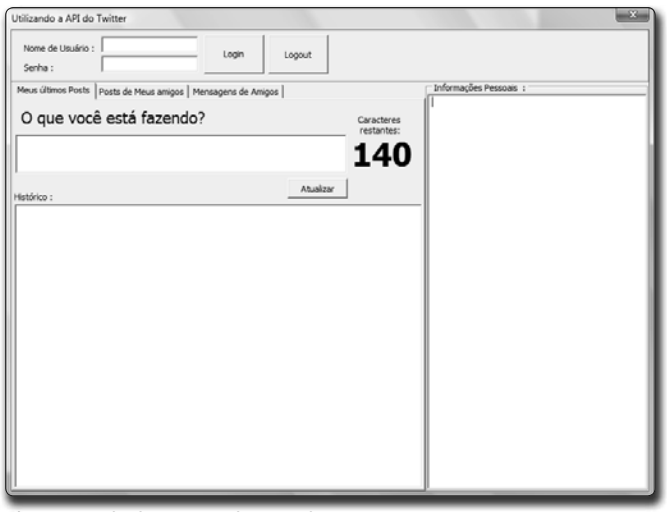


Figura 4. Tela da nossa aplicação de testes

Layout pronto partiremos para as configurações. Para o exemplo, utilizaremos duas rotinas. Uma tratará de exibir as informações pessoais do usuário após a autenticação. A outra se encarregará de mostrar as mensagens de acordo com cada uma das abas selecionadas. Declare no escopo *private* de seu formulário a rotina *GetInformacoesPessoais(),* conforme ilustra a **Listagem 2**.

Listagem 2. Implementação da procedure GetInformacoesPessoais

```
01 procedure TfrmPrincipal.GetInformacoesPessoais();
02 var
03   stDescription, stFollowing, stFollowers, stName, stScreenName,
04   stLocation : String;
05   Node : IXMLNode;
06   i : Integer;
07 begin
08   if XMLDocument.Active then
09   begin
10     for i := 0 to XMLDocument.DocumentElement.ChildNodes.
        Count - 1 do
11     begin
12       Node := XMLDocument.DocumentElement.ChildNodes[i];
13       if Node.NodeName = 'name' then
14       begin
15         if not VarIsNull(Node.NodeValue) then
16           stName := Node.NodeValue;
17       end
18       else if Node.NodeName = 'screen_name' then
19       begin
20         if not VarIsNull(Node.NodeValue) then
21           stScreenName := Node.NodeValue;
22       end
23       else if Node.NodeName = 'location' then
24       begin
25         if not VarIsNull(Node.NodeValue) then
26           stLocation := Node.NodeValue;
27       end
28       else if Node.NodeName = 'description' then
29       begin
30         if not VarIsNull(Node.NodeValue) then
31           stDescription := Node.NodeValue;
32       end
33       else if Node.NodeName = 'friends_count' then
34       begin
35         if not VarIsNull(Node.NodeValue) then
36           stFollowing := Node.NodeValue;
37       end
38       else if Node.NodeName = 'followers_count' then
39       begin
40         if not VarIsNull(Node.NodeValue) then
41           stFollowers := Node.NodeValue;
41       end;
42     end;
43     memProfile.Clear;
44     memProfile.Lines.Add('Nome : ' + stName);
45     memProfile.Lines.Add('Localidade : ' + stLocation);
46     memProfile.Lines.Add('Bio : ' + stDescription);
47     memProfile.Lines.Add('Seguidores : ' + stFollowers);
48     memProfile.Lines.Add('Seguindo : ' + stFollowing);
49   end;
50 end;
```

Como podemos observar entre as **linhas 03** e **06** temos a definição das variáveis que iremos utilizar. Repare que foi criada uma variável *Node*,**linha 05**, do tipo *IXMLNode* que será utilizada para identificarmos os nós correspondentes no *XML*. Na **linha 08** é realizado o teste com o componente *XMLDocument*, verificando se o mesmo está ativo. Após a validação, realizamos uma contagem dos nós do *XML* acessando a propriedade *ChildNodes* que já possuirá um conteúdo dentro de si. Entre as **linhas 13** a **42** é realizada a mesma tarefa, apenas verifica-se se o nome do nó em questão é algum dos nós que desejamos exibir informações. Caso este coincida, testamos se o mesmo não é nulo através do método *VarIsNull* e atribuímos às respectivas variáveis. Na **linha 43** o *memProfile* é limpo e posteriormente carregado com as informações pessoais que encontramos, **linhas 44** a **48**.

Nota

Utilizamos o método VarIsNull devido a propriedade NodeValue ser uma OleVariant. Desta forma, conseguimos verificar seu conteúdo de forma preventiva a erros.

Terminada a primeira *procedure* de nosso formulário, faremos o segundo método responsável por mapear os diferentes tipos de mensagens recebidas. Implemente-o também no escopo *private* e de acordo com a **Listagem 3.**

Observando a **Listagem 3**, percebemos a declaração das variáveis entre as **linhas 03** e **05**. Repare que agora criamos duas variáveis do tipo *IXMLNode*, isto porque iremos mapear um nó dentro de outro, como item e subitem, por exemplo. Perceba que entre as **linhas 07** e **21** é verificada a aba ativa em nosso *PageControl* e de acordo com cada aba, a variável *stParametro* é alimentada com um conteúdo diferente. Na **linha 22** temos a montagem da *URL* completa que fará o resgate das informações. Ainda na mesma linha, podemos reparar que há um parâmetro adicional *count=20*. Este parâmetro delimita ao número de *20* exibições por requisição, note na documentação que existem outros parâmetros, como de paginação por exemplo.

Nas próximas linhas indicamos que será realizada a autenticação através do usuário e senha, **linhas 23** a **25**. Nosso *XMLDocument* então é fechado e carregado com o conteúdo retornado pela *URL* através do método *Get* do componente *IdHTTP*, conforme as **linhas 23** a **25**. O próximo procedimento trata-se de ler o *XML* carregado e extrair suas informações identificando-as. Perceba que contamos os nós primários, e verificamos se um nó primário contém o nó identificado como *user*, através do método *FindNode*, **linhas 30** a **33**. Os valores dos nós são carregados nas variáveis *Post* e *Usuario* simultaneamente, **linhas 34** e **35**. Por último apenas exibimos o conteúdo das variáveis nos respectivos *Memos* de cada uma das páginas da *PageControl*, **linhas 36** a **50**. Em caso de erro, o bloco de isolamento de erros (*except*) será acionado e a mensagem de erro será exibida (**linhas 52** a **58)**.

Após implementarmos os dois métodos principais de nosso artigo, iremos codificar o evento *OnClick* do *btnAtualizarMeusPosts*. Para isto faça-o como ilustra a **Listagem 4**.

Acompanhando a **Listagem 4,** temos a rotina que realiza o envio das informações para o *Twitter*. Antes de qualquer coisa, é necessário verificar se existe conteúdo a ser postado, conforme visto na **linha 06,** caso contrário o compilador finaliza seu curso de execução através do método *Exit* (**linha 07**). Havendo conteúdo, nossa variável *stURL*, responsável pelo envio do *post*, recebe a concatenação das informações de *login*, *senha* e conteúdo a ser postado, **linha 12**. Perceba que na **linha 13** temos a criação de uma *StringList* para a passagem de parâmetro no método *Post*. Entre as **linhas 14** e **16** temos a preparação da autenticação realizada pelo componente *IdHTTP*. Note que na **linha 17** realizamos uma substituição dos espaços pelo caractere *%20*, para que a *URL* seja enviada corretamente.

Finalizando, na **linha 18** temos o envio de informações através do método *Post*, em seguida o *Memo* é limpo e as mensagens são atualizadas através do nosso método anteriormente criado

Listagem 3. Implementação da procedure GetMensagens

```
01 procedure TfrmPrincipal.GetMensagens();
02 var
03   stURL, Usuario, Post, stParametro : String;
04   i : Integer;
05   Node, NodeAux : IXMLNode;
06 begin
07   if pclControle.ActivePage = tbsMeusPosts then
08   begin
09     stParametro := 'user_timeline/';
10     memHistorico.Clear;
11   end
12   else if pclControle.ActivePage = tbsAmigosPosts then
13   begin
14     stParametro := 'friends_timeline/';
15     memFriendPost.Clear;
16   end
17   else if pclControle.ActivePage = tbsMensagensDeAmigos then
18   begin
19     stParametro := 'replies/';
20     memDirectMsg.Clear;
21   end;
22   stURL := 'http://twitter.com/statuses/' + stParametro +
      edtUsuario.Text
       + '.xml?count=20';
23   IdHTTP.Request.BasicAuthentication := True;
24   IdHTTP.Request.Username := edtUsuario.Text;
25   IdHTTP.Request.Password := edtSenha.Text;
26   try
27     XMLDocument.Active   := False;
28     XMLDocument.XML.Text := IdHTTP.Get(stURL);
29     XMLDocument.Active := True;
30     for i := 0 to XMLDocument.DocumentElement.ChildNodes.
        Count - 1 do
31     begin
32       Node    := XMLDocument.DocumentElement.ChildNodes[i];
33       NodeAux := Node.ChildNodes.FindNode('user');
34       Post    := Node.ChildValues['text'];
35       Usuario := NodeAux.ChildValues['screen_name'];
36       if pclControle.ActivePage = tbsMeusPosts then
37       begin
38         memHistorico.Lines.Add('(' + Usuario + ') ' + Post);
39         memHistorico.Lines.Add('--------------------------------')
40       end
41       else if pclControle.ActivePage = tbsAmigosPosts then
42       begin
43         memFriendPost.Lines.Add('(' + Usuario + ') ' + Post);
44         memFriendPost.Lines.Add('--------------------------------')
45       end
46       else if pclControle.ActivePage = tbsMensagensDeAmigos then
47       begin
48         memDirectMsg.Lines.Add('(' + Usuario + ') ' + Post);
49         memDirectMsg.Lines.Add('--------------------------------')
50       end
51     end;
52   except
53     on e: exception do
54     begin
55       ShowMessage(e.Message);
56       exit;
57     end;
58   end;
59 end;
```

Listagem 4. Evento OnClick do btnAtualizarMeusPosts

```
   // Adicione a Unit StrUtils ao Uses do formulário
01 procedure TfrmPrincipal.btnAtualizarMeusPostsClick(Sender: TObject);
02 var
03   stURL : String;
04   StringList : TStrings;
05 begin
06   if Trim(memPostar.Text) = '' then
07     Exit
08   else
09   begin
10     try
11       try
12         stURL := 'http://' + edtUsuario.Text + ':' + edtSenha.Text   +
           '@twitter.com/statuses/update.xml?status=' + memPostar.Text;
13         StringList := TStringList.Create;
14         IdHTTP.Request.Username := edtUsuario.Text;
15         IdHTTP.Request.Password := edtSenha.Text;
16         IdHTTP.Request.BasicAuthentication := True;
17         stURL := AnsiReplaceText(stURL, ' ', '%20');
18         IdHTTP.Post(stURL, StringList);
19         memPostar.Clear;
20         GetMensagens();
21       except
22         on E: exception do
23           ShowMessage('Erro : ' + e.Message);
24       end;
25     finally
26       FreeAndNil(StringList);
27     end;
28   end;
29 end;
```

(**linhas 19** e **20)**. Caso ocorra algum erro, novamente exibiremos uma mensagem que o descreve **linha 23**. Por último, o método *FreeAndNil* é executado, destruindo e liberando o *StringList* da memória, **linha 26.**

Selecione o componente *MemPostar* e defina sua propriedade *MaxLength* para *140,* em seu evento *OnChange* implemente o código que exibe a quantidade restante de caracteres conforme mostra a listagem a seguir:

```
lblContagem.Caption := IntToStr(memPostar.MaxLength -
  Length(memPostar.Lines.Text));
```

Faremos agora a rotina de *Login*, ao conectar-se iremos apenas exibir suas informações pessoais e exibir seus *Post* iniciais. Para isso implemente o código exibido na **Listagem 5** no evento *OnClick* do botão *btnLogin.*

Listagem 5. Evento OnClick do btnLogin

```
01 procedure TfrmPrincipal.btnLoginClick(Sender: TObject);
02 begin
03   IdHTTP.Request.Username := edtUsuario.Text;
04   IdHTTP.Request.Password := edtSenha.Text;
05   IdHTTP.Request.BasicAuthentication := True;
06   XMLDocument.Active := False;
07   XMLDocument.XML.Text := IdHTTP.Get('http://twitter.com/
      users/show/'+
       edtUsuario.Text);
08   XMLDocument.Active := True;
09   GetInformacoesPessoais();
10   GetMensagens();
11   btnLogin.Enabled := False;
12 end;
```

Como podemos observar na **Listagem 5,** o código é bem simples. Entre as **linhas 03** e **05** é realizada a autenticação por meio do componente *IdHTTP*. Em seguida desativamos nosso *XMLDocument,* atribuímos a este o conteúdo da *URL* obtida através do método *Get* e em seguida o ativamos (**linhas 06** a **08)**. Os métodos que trazem as informações pessoais e últimos *posts* são executados e o próprio botão é desabilitado (**linhas 09** a **11)**.

Iremos executar os métodos de exibição de mensagens verificando se o usuário está conectado ou não, portanto, nos eventos *OnClick* dos botões *btnAtualizarPostsAmigos* e *btnAtualizarMsgAmigos* e no evento *OnShow* da aba *tbsMeusPosts* faça o código da listagem que se segue:

```
if not btnLogin.Enabled then
  GetMensagens();
```

Por último faça o evento *OnClick* do *btnLogout*:

```
btnLogin.Enabled := True;
```

Assim finalizamos toda nossa implementação. O resultado final você confere na **Figura 5** que exibe o teste de nossa aplicação em funcionamento.

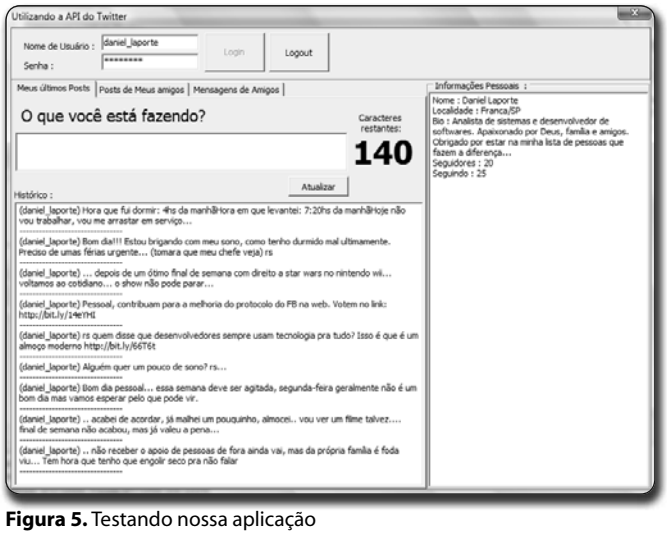


Figura 5. Testando nossa aplicação

Conclusão

Como pudemos perceber, a utilização da API do *Twitter* é bem simples. Verificando os exemplos e documentação disponíveis no *wiki* é possível criar utilitários bem interessantes e dos mais variados tipos. Além disto, podemos utilizar de criatividade para integrarmos estes tipos de recursos à nossas aplicações. Desta forma, podemos até criar canais de comunicação entre clientes, informando-os de possíveis atualizações, manutenções ou outros recursos diversos. Ficamos conectados aos nossos clientes, e eles, aos deles. Ficamos a par do que querem, do que precisam, temos divulgação e feedback imediato. Podemos criar ferramentas específicas para o micro-blog, automatizadas. E graças à API do *Twitter,* podemos fazer tudo isso integrado às nossas atuais aplicações Delphi. Um grande abraço e até a próxima edição. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos relacionados a análise, modelagem, projeto, testes, entre outros temas relacionados a ES

Você testa seus aplicativos de forma completa?

Veja como aplicar em Delphi a prática de Code Coverage para melhorar seus testes

Resumo do DevMan

A realização de testes em sistemas informatizados sempre foi uma tarefa árdua. Quem não aplica alguma forma automatizada de testes acaba por fazer tudo manualmente, demandando assim mais tempo para verificar os diversos cenários que uma determinada rotina pode possuir e com isso algo pode ficar sem ser testado. Quem aplica os testes automatizados, sejam unitários ou de interface, também pode se perguntar se seus testes estão cobrindo todo o código. Para ajudar nessas incertezas, existem aplicativos que verificam o quanto do código foi testado ou executado em um dado momento. Para Delphi temos a ferramenta Discover, usada neste artigo.

Do que trata o artigo

Esclarecer o conceito de Code Coverage e como aplicá-lo em sistemas Delphi, mesmo em sistemas que não utilizam orientação a objetos. Para os desenvolvedores que aplicam testes automatizados com DUnit, poderá ser visto como detectar se seu teste realmente cobre 100% de seu código. Tudo isso será possível utilizando a ferramenta Discover.

Para que serve

Descobrir quais linhas de código de um determinado sistema são executadas em uma determinada rotina, permitindo assim um melhor controle do teste ou depuração.

Em que situação o tema é útil

Muitas vezes clientes nos dizem que determinada rotina apresenta problemas, porém ao realizarmos testes locais em nossas máquinas, o erro não acontece e retornamos a conhecida resposta: aqui está funcionando. Por que será que em seu micro não ocorreu o erro? Será porque não conseguimos reproduzir o ambiente do cliente e com isso alguma parte do código não foi executada? Muito bem, o Code Coverage permite que detectemos esse código que não foi executado mais facilmente e dessa forma, identificar possíveis erros de codificação.

Paulo Quicoli
pauloquicoli@gmail.com
Editor técnico das revistas ClubeDelphi, .Net Magazine e WebMobile Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos, tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com

Code Coverage, ou cobertura de código, é uma medida utilizada em testes de software que possui a finalidade de informar o quanto de código fonte de uma aplicação foi testado. Existem vários tipos de coberturas que podem ser verificadas e cada uma responde uma pergunta. Entre elas temos:

• Cobertura de funções: Cada função do código foi executada?

• Cobertura de decisão: Cada estrutura de controle, por exemplo, cada *if* foi verificado com *true* e *false*?
• Cobertura de caminho: Todas as rotas possíveis de um código foram testadas?
•Cobertura de sentença: Cada linha do código-fonte foi executada?

Observe que realmente essas perguntas são pertinentes. Quem desenvolve seus testes manualmente, executando o sistema e simulando as mais diversas situações, não possui esse tipo de controle, principalmente porque, salvo casos raros, nenhuma documentação formal é gerada disso.

Para ajudar a resolver esse problema, quero introduzir a ferramenta *Discover* que aplica a cobertura de sentença (*statement coverage*) em programas feitos em *Delphi*, oferecendo meios de visualização das linhas que foram executas, produzindo documentação que informa a porcentagem de código executada e até mesmo permite que esse estado seja salvo para uma posterior comparação.

Instalando a ferramenta

O primeiro passo é fazer seu *download*, no *link* localizado no fim do artigo. Sua instalação é bem simples, praticamente não há um instalador. É só descompactar o arquivo para uma determinada pasta e ele está pronto para ser utilizado. Para termos acesso rápido ao *Discover* é possível instalá-lo no menu *Tools* do *Delphi*. Para isso acesse, no *Delphi*, o menu *Tools>Configure tools*. Na janela que surgir escolha *Add* e configure para que os caminhos solicitados apontem para a pasta onde está localizado o executável da ferramenta, como mostra a **Figura 1**.

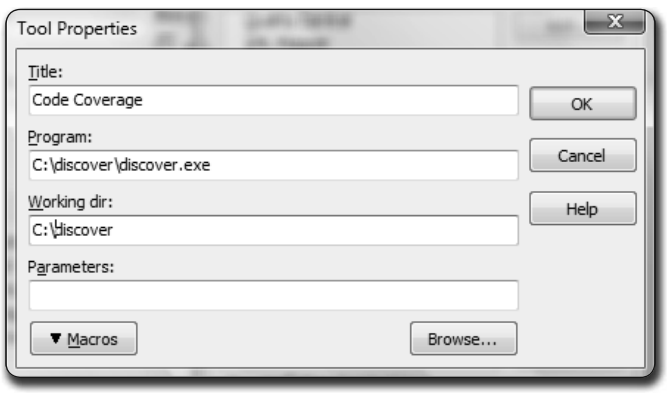


Figura 1. Instalando o Discover no menu Tools

Criando um programa a ser verificado

Para podermos testar seus recursos é preciso ter um aplicativo a testar. Por isso criei um pequeno projeto *VCL Forms Application* no *Delphi 2010* que exibe uma caixa de diálogo baseando-se em três *RadioButtons*. Na **Figura 2** vemos sua tela principal. Na **Listagem 1** temos o código do botão *Chamar Diálogo*. Como pode ser observado, apenas verificamos qual opção está marcada e exibimos a mensagem adequada.

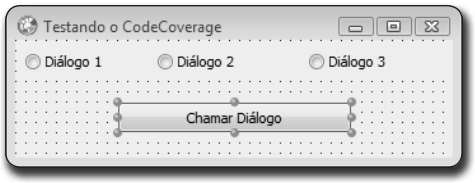


Figura 2. Programa a ser testado

Listagem 1. Evento OnClick

```
procedure TForm1.btChamarClick(Sender: TObject);
begin
  if rbDialogo1.Checked then
     ShowMessage('Sou o diálogo 1')
  else
     if rbDialogo2.Checked then
        ShowMessage('Sou o diálogo 2')
     else
        if rbDialogo3.Checked then
            ShowMessage('Sou o diálogo 3');

end;
```

Realizando o code coverage

Para que o *Discover* possa verificar quais linhas de código são executadas é necessário primeiramente ativar a geração do arquivo *Map* do *Delphi*, em nível detalhado, durante a compilação do projeto. Para isso acesse o menu *Project>Options>Linking* e ative a opção exibida na **Figura 3**.

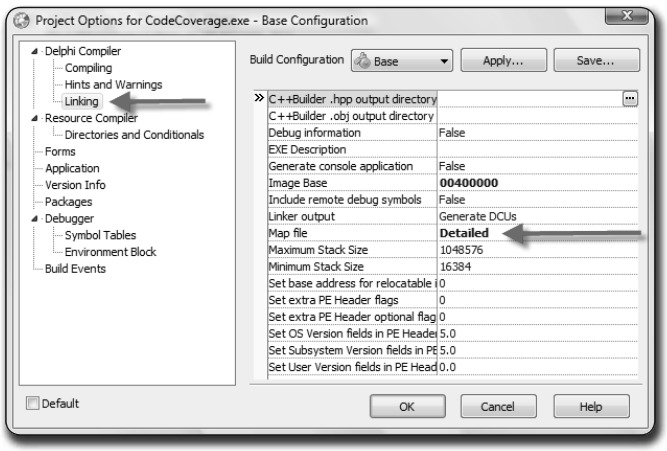


Figura 3. Habilitando a geração do arquivo Map

Compile o projeto e logo após acesse o menu *Tools* e escolha o item *CodeCoverage*. A ferramenta *Discover* será aberta. Nela escolha o menu *Project>Load* e localize o arquivo de projeto de nossa aplicação. Ao carregar o projeto o mesmo já passa por uma avaliação e nela temos algumas informações que estão organizadas nas guias *Summary, Units, Routines* e *Overview*. Vamos entender o que cada guia informa. Na **Figura 4** temos a primeira guia.

Nesta parte podemos ter um resumo da aplicação organizado em *units, routines, points* e *coverage*. Em *units* temos:
• **Total:** é o número total de *units Delphi* envolvidas na aplicação;
• **Source:** é o número de *units* com código-fonte disponível que será utilizado pelo *Discover*;
• **Foreground:** é o número de *units* que pode ser analisado pelo *Discover*.

Nota do DevMan

O Delphi pode gerar arquivos de debug que contêm detalhes de erros bem como o código que levantou o erro. Esse é o arquivo Map. Ele contém os endereços de memória e os números de linhas agrupadas por unit. A seguir vemos parte do arquivo Map para o programa de exemplo criado.

```
Line numbers for Unit1(Unit1.pas) segment .text

  30  0001:000B30FC       31  0001:000B30FF       32  0001:000B3111       34
      0001:000B311D
  35  0001:000B312F       37  0001:000B313B       38  0001:000B314D       40
      0001:000B3157
  42 0001:000B31E7

Line numbers for CodeCoverage(CodeCoverage.dpr) segment .itext

   9  0002:00000C48       10  0002:00000C58       11  0002:00000C64       12
      0002:00000C72
  13   0002:00000C8A    14 0002:00000C96

Bound resource files

c:\program files\embarcadero\rad studio\7.0\lib\character.res
c:\program files\embarcadero\rad studio\7.0\lib\Controls.res
Unit1.dfm
CodeCoverage.res
CodeCoverage.drf

Program entry point at 0002:00000C48
```

O *Discover* utiliza o conceito de *Foreground* e *Background* para identificar quais units podem ser analisadas. *Units* em *background* são aquelas do próprio *Delphi*, por exemplo a *VCL*, que não precisa ser verificada. *Foreground* são as *units* do seu sistema. Continuando, na **Figura 5** temos a guia *Units*.

Aqui temos o nome de cada *unit* envolvida no projeto. O tamanho (*size),* quantidade de rotinas (*R. Qty),* quantidade de rotinas com cobertura 0 (*R.0%),* quantidade de rotinas com 100% de cobertura (*R.100%*) e a porcentagem de cobertura (*coverage)* de cada uma é exibida. Os ícones indicam a situação de cada *unit*. Um ícone com o *X* em vermelho indica que não foi encontrado o código-fonte ou o código-fonte não possui rotinas a serem testadas. O ícone com apenas um *livro fechado* indica que essa é uma *unit background*. O ícone de *livro aberto* representa uma *unit foreground*. A próxima guia é a *Routines*, que pode ser vista na **Figura 6**.

Nesta guia temos a rotinas que serão verificadas. Veja que ao clicar no nome da rotina seu código é exibido ao lado com as marcações que são chamadas de *points* (pontos). Esses pontos são os trechos de código que serão analisados pelo *Discover*. Caso alguma rotina não precise ser analisada, basta clicar com o botão direito do *mouse* sobre seu nome, na guia *routines* e enviá-la para *background*.

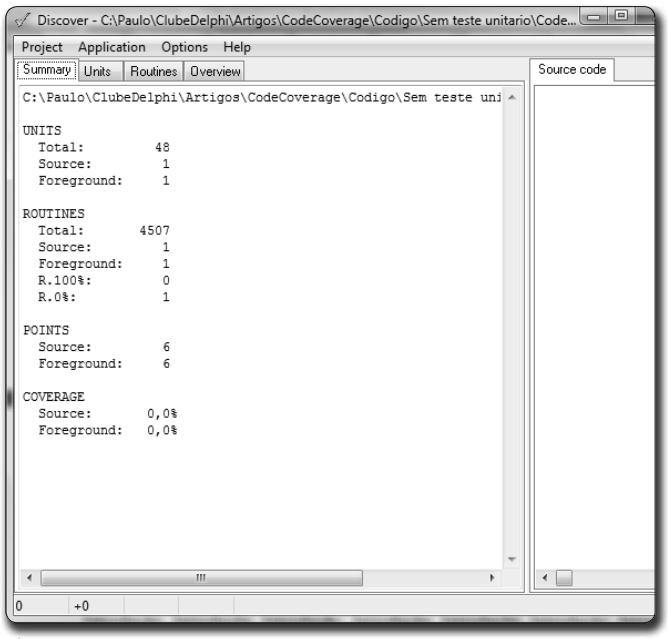


Figura 4. Guia Summary

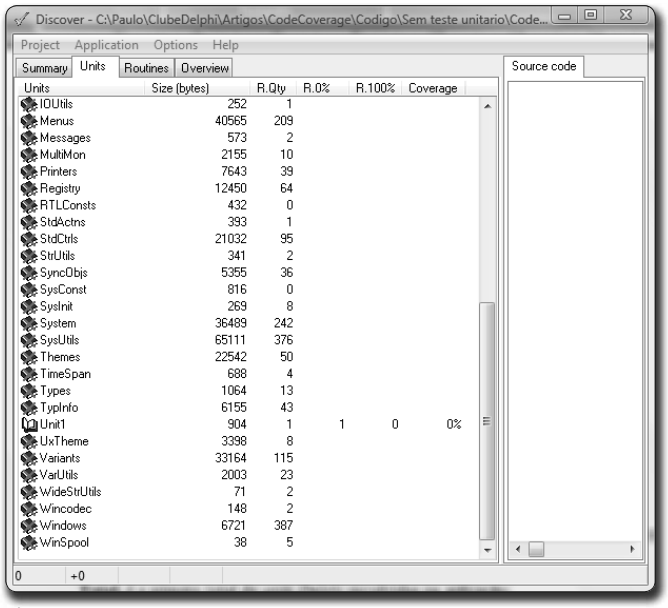


Figura 5. Guia Units

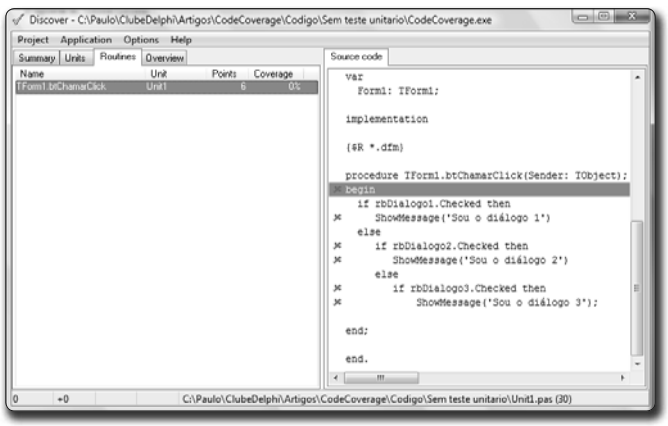


Figura 6. Guia Routines

A última guia, *Overview*, exibe uma representação gráfica do percentual de cobertura de cada rotina que compõe o sistema. Como no nosso caso apenas uma rotina será testada, apenas um quadrado é exibido, como mostra a **Figura 7**.

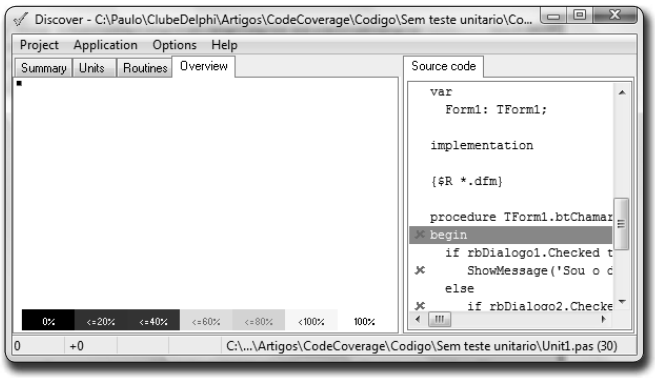


Figura 7. Guia Overview

Agora que entendemos o que é exibido pelo *Discover*, vamos realizar a análise que acontece em tempo de execução. Clique no menu *Application>Run*. Nossa aplicação de demonstração é executada. Clique agora no primeiro *RadioButton* e clique no botão. Observe em seguida como fica o painel *Source Code* do *Discover*. Ele deve estar como na **Figura 8**.

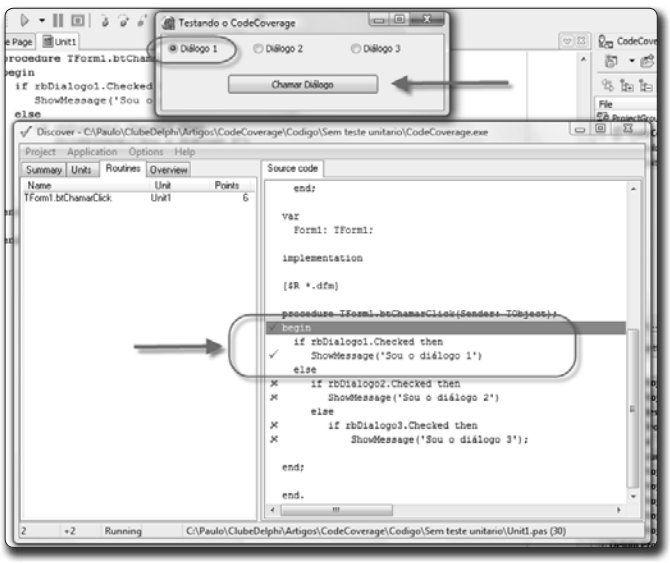


Figura 8. Executando a análise

Como pode ser visto, automaticamente ao executar a aplicação podemos ter um *feedback* de quais linhas de código são executadas. Se voltarmos para a guia *Units* saberemos o percentual de execução da rotina *TForm1.btChamarClick*, como pode ser visto na **Figura 9**.

Se continuarmos a executar a aplicação escolhendo os outros dois itens veremos os códigos sendo executados. Para quem não utiliza testes automatizados isso pode ser muito útil, pois veremos na hora o que é executado, além de poder manter isso documentado para posteriores comparações.

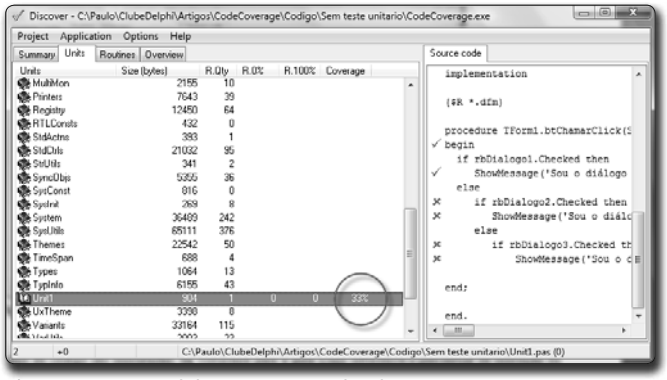


Figura 9. Percentual de execução atualizado

Utilizando o Discover junto com testes unitários

Podemos utilizar o *Discover* junto com testes unitários para saber se os testes estão verificando cada parte do código. Isso é muito importante para que tenhamos certeza que tudo foi testado. Vamos criar um novo projeto de teste. Neste vamos simular uma tela de consulta de contatos, onde caso o nome informado não seja encontrado, um aviso deve ser dado e também, caso o nome seja inválido (vazio), um alerta seja exibido.

O novo aplicativo a ser testado

Após criar uma nova aplicação do tipo *VCL Forms*, um *DataModule* foi adicionado. Nele temos um *ClientDataSet* que será utilizado como fonte de dados para nossa consulta e um *DataSource* será ligado a ele. Nesse *ClientDataSet* temos apenas um campo (*Nome, string, 100)*. Isso pode ser visto na **Figura 10**.

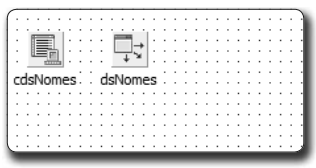


Figura 10. DataModule

Nota

Nota: O código-fonte dos exemplos aqui demonstrados está disponível para download. Não entrei em detalhes de como o programa exemplo foi criado para assim mantermos o foco no uso da ferramenta Discover.

Vamos criar alguns dados fictícios para que possamos realizar a busca. Criamos uma *procedure* chamada *CriaDados* no *DataModule* e em seu evento *OnCreate* a executamos, como mostra a **Listagem 2**.

No formulário principal montamos uma tela de consulta com um *TEdit*, um *TButton* e um *TDBGrid*, como exibe a **Figura 11**.

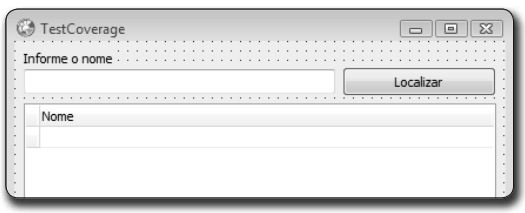


Figura 11. Formulário principal

Listagem 2. Criando os dados para consulta

```
procedure Tdtm.CriaDados;
begin
  cdsNomes.CreateDataSet;
  cdsNomes.Append;
  cdsNomesNome.AsString := 'Paulo Roberto';
  cdsNomes.Append;
  cdsNomesNome.AsString := 'Ana Paula';
  cdsNomes.Append;
  cdsNomesNome.AsString := 'Cesar Augusto';
  cdsNomes.Append;
  cdsNomesNome.AsString := 'Maria Rita';
  cdsNomes.Post;
end;

procedure Tdtm.DataModuleCreate(Sender: TObject);
begin
  CriaDados;
end;
```

Nesse momento nosso aplicativo possui duas camadas lógicas: a de apresentação, constituída pelo formulário principal e dados, centralizada no *DataModule*. Vamos criar agora a camada de negócios que vai ser responsável em realizar a busca propriamente dita. Inserimos uma nova *unit* chamada *controllerU*, inspirada no padrão *MVC* e nela definimos e implementamos a classe *TConsulta*. Veja **Listagem 3**.

Listagem 3. Definição e implementação da classe controller

```
type
  EnumRetornoBusca = (rbNaoEncontrado, rbBuscaInvalida, rbEncontrado);

  TConsulta = class
  public
    function BuscarNome(Nome:string): EnumRetornoBusca;

  end;

implementation

{ Consulta }

function TConsulta.BuscarNome(Nome: string): EnumRetornoBusca;
begin
   if Trim(Nome)='' then
   begin
     result := rbBuscaInvalida;
     exit;
   end;
   dtm.cdsNomes.First;
   if dtm.cdsNomes.Locate('Nome',Nome,[loPartialKey]) then
   begin
      result := rbEncontrado
   end
   else
   begin
     result := rbNaoEncontrado;
     dtm.cdsNomes.First;
   end;
end;
```

Observe que definimos logo no início uma enumeração chamada *EnumRetornoBusca*. Esta será utilizada para identificar o resultado da consulta. Então temos o método *BuscarNome*. Nele verificamos se o parâmetro de busca passado é válido. Se não for o retorno do método indica isso através do valor *rbBuscaInvalida*. Caso seja válido, a busca é feita retornando o valor de acordo com seu resultado.

Vamos agora utilizar esse *controller* na camada de apresentação, justamente no evento *OnClick* do botão de consulta, como mostra a **Listagem 4**.

Aqui nós instanciamos o *controller* e executamos seu método *BuscarNome*. O resultado é então avaliado em uma estrutura *case..of* para sabermos como exibir as mensagens de notificação ao usuário. Vamos criar um teste agora utilizando o assistente existente no *Delphi 2010*.

Criando o teste unitário

Os testes unitários, como já dizem o nome, visam testar de forma automática partes pequenas de aplicativos, mais precisamente métodos de classes. São métodos que testam outros métodos. Vamos testar se o método *BuscarNome* da classe *TConsulta* está funcionando de acordo e se o teste vai cobrir 100% desse método.

Utilizando o assistente do *Delphi 2010* para criar os testes, que utiliza o *framework DUnit* que é instalado junto com o *Delphi*, acessamos o menu *File>New>Other* e escolhemos a opção *Unit Test>Test Project*. Na tela que surgir, informamos um nome ao projeto e avançamos. Na próxima tela escolhemos a opção *GUI* para o item *Test Runner* e finalizamos. Com isso vamos poder rodar nossos testes e ver o resultado de forma gráfica, em janelas e não como em uma aplicação do tipo console.

Nota do DevMan

MVC (Model-View-Controller) é um padrão de arquitetura criado com o objetivo básico de facilitar a construção e manutenção de softwares separando a camada de dados da camada de visualização. Para isso ela define três elementos que compõem o padrão. São eles:

Model – Contém a regra de negócio e representa o domínio da aplicação;
View – Responsável por exibir as informações para o usuário;
Controller – Realiza interação entre View e Model.

Para saber mais sobre esse padrão e como aplicá-lo em Delphi, confira a edição 90.

Nota do DevMan

Existe um estilo de programação chamado TDD (Test Driven Development) onde se recomenda fazer o teste unitário antes de se desenvolver as classes e métodos. Como as classes e métodos não foram implementados, todos os testes falharão. Assim, depois de se desenvolver os métodos necessários, rodam-se novamente os testes e se verifica o resultado. Saiba mais sobre testes unitários na edição 104 e mais sobre TDD na edição 78.

Listagem 4. Invocando o controller

```
procedure TForm1.btnLocalizarClick(Sender: TObject);
var
  controller: TConsulta;
begin
  controller := TConsulta.Create;
  case controller.BuscarNome(editLocalizar.Text) of
    rbNaoEncontrado: ShowMessage('Nome não encontrado');
    rbBuscaInvalida: ShowMessage('O nome a se buscar não pode
     ser vazio');
  end;
  FreeAndNil(controller);
end;
```

Veja que um novo projeto foi criado. Adicione a ele os arquivos do projeto de consulta, *dtmU.pas* e *controllerU.pas*. Agora adicione uma nova *unit* e a deixe como mostra a **Listagem 5**.

Vamos a uma breve explicação dessa listagem. Como já foi mencionado um teste unitário nada mais é do que um código que testa outro código. Portanto centralizamos nossos testes em *TestCases*, que são uma forma de organizá-los. Nosso *TestCase* (caso de teste) é representado pela classe *ConsultaTestCase*. Nela temos uma referência ao *controller*, dois métodos protegidos e um método publicado.

Os métodos protegidos *SetUp* e *TearDown* são definidos pelo framework *DUnit*. Eles respectivamente são chamados antes da execução de um teste e logo após seu término. São utilizados para preparar o ambiente do teste e para desalocar recursos utilizados pelo teste. Observe a implementação do método *SetUp*. Nele instanciamos o *DataModule* e instanciamos o *controller* que faz as buscas. No *TearDown* apenas liberamos da memória tudo o que foi criado anteriormente.

Quanto ao método publicado *Consulta*, ele é o teste em si. Veja seu código e note que estamos verificando o retorno do método *ConsultaNome* através do comando *Check*. Esse comando está disponível no *framework DUnit* e é utilizado para verificar se a condição informada é verdadeira.

Neste momento salve o projeto e compile.

Listagem 5. Implementação dos testes

```
unit Testes;

interface

uses TestFramework,
     SysUtils,
     DtmU,
     controllerU;

implementation

type
  ConsultaTestCase = class(TTestCase)
  private
    controller: TConsulta;
  protected
    procedure SetUp; override;
    procedure TearDown; override;
  published
    procedure Consulta;


  end;

{ ConsultaTestCase }

procedure ConsultaTestCase.Consulta;
begin
   Check(controller.BuscarNome('Não Encontrado')=rbNaoEncontrado);
   Check(controller.BuscarNome('Paulo')=rbEncontrado);
end;

procedure ConsultaTestCase.SetUp;
begin
  dtm := TDtm.Create(nil);
  controller := TConsulta.Create;
end;

procedure ConsultaTestCase.TearDown;
begin
  FreeAndNil(controller);

  dtm.cdsNomes.Close;
  FreeAndNil(dtm);
end;

initialization
  TestFramework.RegisterTest(ConsultaTestCase.Suite);
```

Verificando a cobertura do teste

Como fizemos anteriormente, vamos fazer uso da ferramenta *Discover* para saber se nossos testes estão cobrindo 100% de nosso código. Vamos então habilitar a geração do arquivo *MAP*, acessando o menu *Project>Options>Linking*, como já mostrou a **Figura 3**.

Então abrimos o aplicativo *Discover* pelo menu *Tools>Code Coverage* do *Delphi*, criado anteriormente. No menu *Project>Load* do *Discover* adicionamos o arquivo *.DPR* do nosso teste unitário e então selecionamos *Application>Run*. Nosso teste é exibido aguardando nossa confirmação para ser executado, como mostra **Figura 12**.

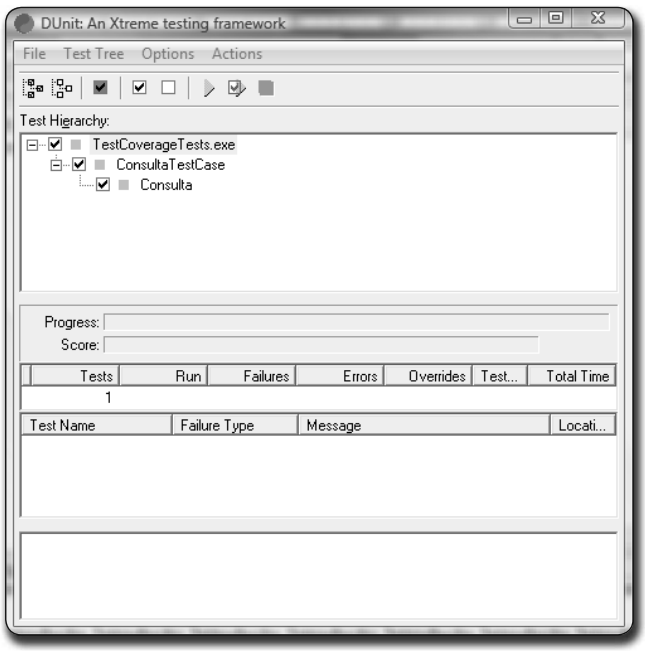


Figura 12. DUnit em execução

Clique agora no botão verde, que parece um botão *play* e o teste será realizado. Após sua execução feche para que voltemos à tela do *Discover*. Nele vamos acessar a guia *Routines* para ver se todos os métodos foram 100% executados (**Figura 13**).

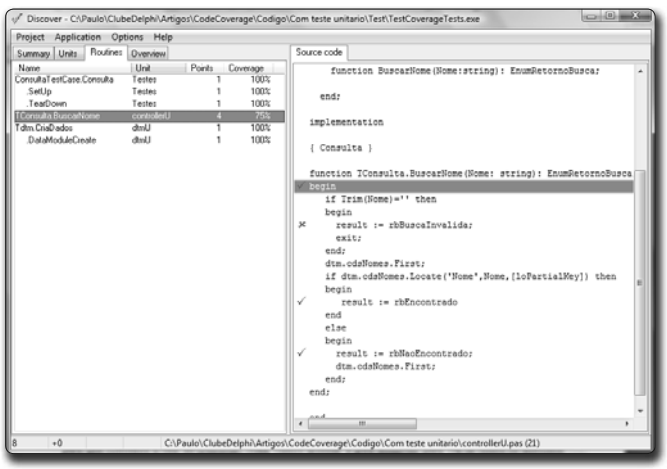


Figura 13. Verificando a execução dos métodos na Guia Routines

Observe agora que o método *TConsultaBuscarNome* não foi executado totalmente, mas sim, parcialmente (75%), ao lado, na guia *Source code* podemos ver qual parte não foi executada pelo nosso *TestCase ConsultaTestCase.Consulta*. Vamos rever esse código a seguir:

```
procedure ConsultaTestCase.Consulta;
begin
   Check(controller.BuscarNome
    ('Não Encontrado')=rbNaoEncontrado);
   Check(controller.BuscarNome('Paulo')=rbEncontrado);
end;
```

Veja, realmente estamos testando apenas dois retornos do método *BuscarNome*, embora ele possa retornar três. Aqui podemos entender a real importância do *Code Coverage*. Com ele nós fomos capazes de identificar o que está faltando em nosso teste. Vamos arrumá-lo, deixando-o como o seguinte código:

```
procedure ConsultaTestCase.Consulta;
begin
   Check(controller.BuscarNome
    ('Não Encontrado')=rbNaoEncontrado);
   Check(controller.BuscarNome('Paulo')=rbEncontrado);
   Check(controller.BuscarNome('')=rbBuscaInvalida
    );
end;
```

Recompile o projeto de teste, reabra e execute no *Discover*, como fizemos anteriormente. Veja agora na **Figura 14** o resultado de nossa correção. Agora sim, nosso método *BuscarNome* foi completamente testado.

Conclusão

Testar *software* é uma tarefa árdua e o uso de ferramentas que possam trazer mais segurança nesse trabalho é sempre bem-vindo. Existem outras formas de se testar um aplicativo, como ferramentas que simulam a utilização do sistema por uma pessoa. Todas essas formas de se testar, mas especialmente os testes unitários, estão aí para nos dar confiança ao realizar alterações no código de nossos sistemas. Pois, como tudo é automatizado, fica fácil detectar alguma alteração que venha causar efeitos colaterais no sistema todo. ●

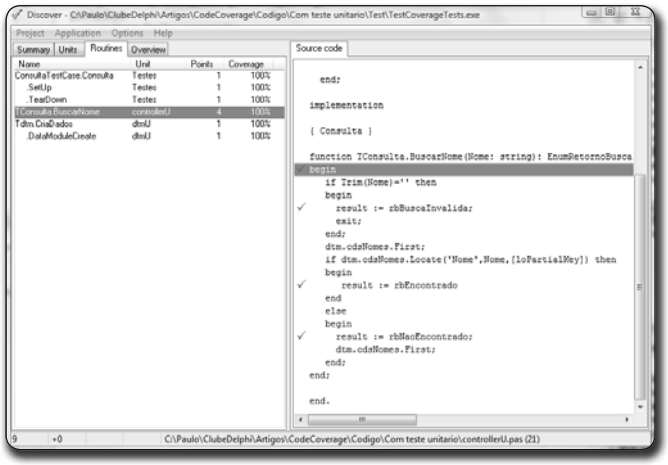


Figura 14. Análise do Discover

Links

Download da ferramenta Discover

http://www.cyamon.com

Download do framework DUnit

http://dunit.sourceforge.net

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

RTTI

Conheça as novidades na RTTI do Delphi 2010

Rodrigo Carreiro Mourão

rodrigocarreiro@rmfactory.com.br

Apaixonado por tecnologia da informação. Atualmente é sócio diretor da RM Factory Soluções em TI, empresa especializada em consultoria na área de TI e desenvolvimento de softwares customizados, parceira da TDS Tecnologia - RJ. Possui os títulos de BDS2006 Win32 Product Certified e Borland Instructor Certified. Editor Técnico da Revista Clube Delphi e Coordenador da área de Delphi no sistema de suporte on-line da DevMedia. Palestrante da Borland Conference 2007. Mantenedor da comunidade www.delphisophp.com, a primeira do Brasil 100% feita com Delphi for PHP. Autor de diversos artigos técnicos veiculados nos principais sites e revistas sobre Delphi do Brasil .

Do que trata o artigo

Reflexão de objetos. Toda classe deve saber quais são suas propriedades, métodos, campos e atributos. A reflexão, que no Delphi é chamada de RTTI, é um framework interno da ferramenta de desenvolvimento que disponibiliza meios para obtenção de tais informações. Outras plataformas de desenvolvimento, como o .NET e Java, possuem um framework de reflexão muito rico, e só agora, na versão 2010, o Delphi traz recursos melhorados em sua RTTI.

Para que serve

Através da RTTI podemos obter todos os métodos e propriedades, com seus valores, de uma classe instanciada. Por classe entenda-se as suas, que você pode vir a criar, ou até mesmo as existentes no Delphi, como TEdit, TClientDataSet. De posse dessas informações, podemos utilizá-las da forma que for necessário, podendo até mesmo alterar seus valores (no caso de propriedades), usando RTTI.

Em que situação o tema é útil

O uso da RTTI está quase sempre ligado a frameworks que venham a ser desenvolvidos para centralização e otimização dos sistemas. Esses frameworks, grandes ou não, tem por objetivo trazer maior flexibilidade a um sistema e produtividade ao desenvolvedor. Um exemplo seria um framework de validação, onde centralizaríamos na classe a ser validada todo o código responsável por isso, ao invés de replicar por todo sistema uma regra de negócio.

Resumo do DevMan

O Delphi sempre ofereceu recursos de reflexão, porém estes eram limitados quando comparados com a reflexão existente no .Net ou Java. Agora, na versão 2010, a RTTI passou por uma grande atualização trazendo benefícios esperados pelos programadores Delphi. Neste artigo vamos entender o que é reflexão e como ela foi implementada no Delphi. Também veremos como utilizar na prática o novo recurso adicionado, chamado Custom Attribute, além de conhecer novas perspectivas de desenvolvimento.

Runtime Type Information (RTTI) é um paradigma de programação em que a informação sobre um tipo pode ser obtido em tempo de execução. As versões anteriores do Delphi já davam suporte à *RTTI,* porém de uma forma limitada. As propriedades deveriam ser colocadas na seção *published*. As classes deveriam ter as diretivas {$M+} para que pudessem ser lidas via RTTI ou herdar de uma classe que a possuísse. Alguns tipos não podiam ser publicados como o Real48, *array*, entre outros.

Sem falar que o compilador não nos dava informações como, por exemplo, quais propriedades foram declaradas no objeto que está sendo lido e quais são herdadas. Pode não parecer, mas esta informação é de extrema importância quando pensamos em **Mapeamento Objeto Relacional**.

A partir de agora tudo isso mudou e para melhor! A RTTI do Delphi 2010 foi toda reformulada e traz agora novos e interessantes recursos como, por exemplo, a possibilidade de inserirmos atributos em nossas classes, propriedades, método etc. Antes de partir para um exemplo prático, vamos examinar a nova unit *RTTI.pas*.

RTTI.pas

A *unit RTTI.pas* traz novos e excelentes tipos que facilitam e muito trabalhar com RTTI. A **Tabela 1** mostra estas principais classes e *records*, com uma breve descrição de sua função e os principais métodos presentes em cada uma delas.

Mão na Massa

Vamos pôr a mão na massa, afinal a experiência é um fruto da prática e não da teoria. Crie uma nova aplicação no Delphi 2010. Adicione uma nova *unit* e salve-a como *uModel.pas*. Nela vamos adicionar uma pequena classe de exemplo, apenas contendo algumas propriedades. A intenção é mapear esta classe e extrair algumas informações. A **Listagem 1** mostra a classe completa.

Não entrarei em detalhes sobre o código da **Listagem 1** pois foge do escopo do artigo, até porque o código é muito simples temos apenas uma classe com duas propriedades (Nome e Idade). Bem até agora nada de novidade. No *form* da sua aplicação, adicione uma *TreeView* e um botão. Na seção private do *form* cria uma *procedure* com a seguinte assinatura:

Listagem 1. Criação da Classe TExemplo

```
unit uModel;

interface

type
  TExemplo = class
  private
    FIdade: Integer;
    FNome: String;
    procedure SetIdade(const Value: Integer);
    procedure SetNome(const Value: String);
  public
    property Nome: String read FNome write SetNome;
    property Idade:Integer read FIdade write SetIdade;
  end;

implementation

procedure TExemplo.SetIdade(const Value: Integer);
begin
  FIdade := Value;
end;

procedure TExemplo.SetNome(const Value: String);
begin
  FNome := Value;
end;

end.
```

```
procedure GetObjectInformation(Obj: TObject);
```

Nota

Declare uModel e RTTI no uses do form.

TRttiContext (Record)	Principal tipo da nova unit, através deste record é que extraímos as informações de um determinado tipo.	
	GetType	Método utilizado para retornar um TRttiType contendo as informações do tipo passado como parâmetro.
TRttiType	Classe utilizada para representar a estrutura de um objeto. Através dos métodos desta classe podemos extrair inúmeras informações de um tipo em tempo de execução.	
	GetMethods	Retorna a lista de métodos do Objeto.
	GetFields	Retorna a lista de Fields do Objeto.
	GetProperties	Retorna a lista de properties do Objeto.
	GetAttributes	Retorna a lista de atributos do Objeto.
TRttiMethod	Classe utilizada para representar a estrutura de um método. Através dos métodos desta classe podemos extrair inúmeras informações de uma procedure ou function em tempo de execução.	
	GetParameters	Retorna a lista de parâmetros do método caso haja
	ReturnType	Em caso de função retorna o tipo do retorno da função
	IsConstructor	Verifica se o método é um construtor
	IsDestructor	Verifica se o método é um destrutor
	GetAttributes	Retorna a lista de atributos do Método
TRttiField	Classe utilizada para representar a estrutura de um Field. Através dos métodos desta classe podemos extrair inúmeras informações de um Field em tempo de execução.	
	GetValue	Retorna um TValue que representa o conteúdo do Field
	FieldType	Representa o tipo do Field em questão (TRttiType)
TRttiProperty	Classe utilizada para representar a estrutura de uma property. Através dos métodos desta classe podemos extrair inúmeras informações de uma property em tempo de execução.	
	PropertyType	Representa o tipo da property em questão (TRttiType)
	GetValue	Retorna um TValue que representa o conteúdo da property

Tabela 1. Classes Disponíveis na unit RTTI.pas

Listagem 2. Procedure GetObjectInformation

```
1.   var
2.   CT: TRttiContext;
3.   Tipo: TRttiType;
4.   Prop: TRttiProperty;
5.   Met: TRttiMethod;
6.    Att: TCustomAttribute;
7.  MainNode, TypeNode, AttNode: TTreeNode;
8.  begin
9.      Tipo := CT.GetType(Obj.ClassType);
10.   MainNode := TreeView1.Items.Add(nil,Obj.ClassName);

11.    TypeNode := TreeView1.Items.AddChild(MainNode,'Propriedades');
12.   for Prop in Tipo.GetProperties do
13.    begin
14.    TreeView1.Items.AddChild(TypeNode,Prop.Name+' -'+Prop.
        GetValue(Obj).ToString);
15.   end;

16.   TypeNode := TreeView1.Items.AddChild(MainNode,'Métodos');
17.    for Met in Tipo.GetMethods do
18.   begin
19.   TreeView1.Items.AddChild(TypeNode,Met.Name);
20.   end;
21.    end;
```

Na implementação desta *procedure* insira o código da **Listagem 2.**

A função da *procedure* descrita na **Listagem 2** é extrair as informações do objeto passado como parâmetro e exibi-las na *TreeView*. Essas informações estarão separadas pelas seguintes categorias: propriedades, métodos e atributos. Nas linhas de 2 a 7 temos a declaração das variáveis que iremos utilizar neste processo. Destaque para as variáveis do *CT, Tipo, Prop, Met e Att*. Todas são de tipos exclusivos da nova estrutura *RTTI*. Todas estas classes já foram suficientemente explicadas anteriormente. Ainda na seção de declaração de variáveis temos a presença de três variáveis *TTreeNode* que servirão para que possamos montar a estrutura da *TreeView*.

Iniciamos o procedimento na linha 9 invocando o método *GetType* para extrairmos todas as informações do objeto passado como parâmetro para a variáveis *Tipo*. Observe que na verdade não passamos o objeto para o método *GetType* e sim o tipo: *Obj.ClassType*.

Na sequência, linha 10, criamos o primeiro nó da nossa *TreeView*, adicionamos como *caption* o nome da classe passada como parâmetro e por fim guardamos o ponteiro para o nó na variável *MainNode*. Feito isso adicionamos um nó filho neste nó principal e colocamos o *caption* propriedades. É sob este nó que adicionaremos todas as propriedades que o objeto passado possui. Nas linhas 12 a 15 temos o código que executa esta tarefa.

Nós fazemos um loop utilizando a estrutura *for in* na lista de propriedades do objeto, linha 12, e para cada propriedade encontrada adicionamos um nó (dentro da propriedade) e nele adicionamos o nome da propriedade mais o seu valor como string, linha 14. Na linha 16 adicionamos um novo nó, desta vez sob o nó principal, com o *caption* métodos. Por fim repetimos o procedimento que utilizamos para recuperar as propriedades, porém desta vez estamos recuperando os métodos, linhas 17 a 20.

Para testarmos a aplicação vá ao evento *OnClick* do botão e insira o código da **Listagem 3**. A **Figura 1** mostra a aplicação em execução.

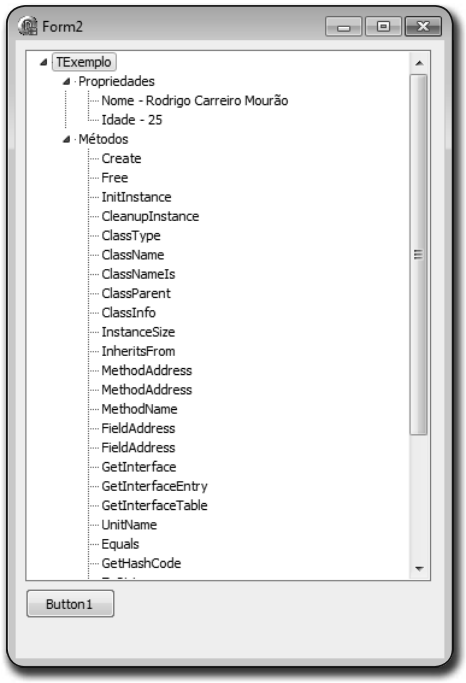


Figura 1. Aplicação em execução

Listagem 3. Código do evento OnClick do botão

```
var
  Ex: TExemplo;
begin
  Ex       := TExemplo.Create;
  Ex.Nome  := 'Rodrigo Carreiro Mourão';
  Ex.Idade := 25;
  GetObjectInformation(Ex);
end;
```

Nota

Você pode estar se perguntando de onde vieram estes métodos. Mas não esqueça de que embora não tenhamos declarado nenhum método em nossa classe, estamos obviamente herdando de TObject, e com isso temos em TExemplo todos os métodos declarados em TObject.

TCustomAttribute

O melhor ainda esta por vir. No exemplo que acabamos de fazer não tocamos sequer no assunto atributos. Mas o que são atributos? São *properties*? São *Fields*? Ou é tudo a mesma coisa?

Quando declaramos uma classe no Delphi geralmente criamos propriedades e métodos. Repare que eu disse propriedades. Propriedades como sabemos são apenas um canal de acesso a um campo que efetivamente guarda a informação propriamente dita. Por ser um canal de acesso as *properties* vem acompanhadas sempre de métodos *Get / Set*, para leitura e escrita, respectivamente. Esse é um conceito de encapsulamento presente na orientação a objetos.

Mas e os atributos o que são? Um atributo é toda e qualquer informação extra que você queira adicionar a uma *property*, *Field*, *procedure*, *function*, classe e etc. Uma *property*, por exemplo, traz apenas seu nome, seu valor e o seu tipo. Suponha que agora você queira adicionar uma atributo como valor máximo ou valor mínimo para esta *property*.

Simples, crie um atributo para valor máximo e um atributo para valor mínimo e adicione a sua *property*. Para criar um atributo você deverá criar uma classe descendente de *TCustomAttribute*, classe básica para todo e qualquer atributo que você queira criar.

Vamos criar um atributo para inserirmos em nossa classe *TExemplo*. Vamos adicionar uma descrição para nossas propriedades. Para isso insira o código da **Listagem 4** na *unit uModel*.

Listagem 4. Criando o atributo Descrição

```
Descricao = class(TCustomAttribute)
private
  FTexto: String;
public
  constructor Create(const Texto: String);
  property Texto: String read FTexto write FTexto;
end;
```

Repare que no código da **Listagem 2** apenas criamos um classe como outra qualquer. Só atente para dois detalhes quando for criar seus atributos:

1. Eles obrigatoriamente devem herdar de *TCustomAttribute;*
2. O construtor deve ter tantos parâmetros quantas propriedades tiverem sua classe.

Agora vem o mais interessante, "decorar" (esse é o termo utilizado quando se aplica o atributo) nossa classe com o mais novo atributo criado: Descrição. A **Listagem 5** mostra como fica nossa classe *TExemplo* usando nosso atributos.

Listagem 5. Classe TExemplo com o uso dos atributos

```
[Descricao('Esta esta é uma classe de Exemplo')]
TExemplo = class
private
  FIdade: Integer;
  FNome: String;
  procedure SetIdade(const Value: Integer);
  procedure SetNome(const Value: String);
public
  [Descricao('Esta propriedade armazena o conteudo do Nome')]
  property Nome: String read FNome write SetNome;
  [Descricao('Esta propriedade armazena o conteudo da Idade')]
  property Idade:Integer read FIdade write SetIdade;
end;
```

Veja a facilidade em se adicionar um atributo a uma propriedade. Nós utilizamos a seguinte sintaxe *[TClasse(parâmetro,parâmetro)]* sempre sobre a estrutura que queremos adicionar o atributo, seja ela uma propriedade, classe ou método.

No nosso exemplo adicionamos uma descrição para cada uma das *properties* que temos em nossa classe. Caso tenha necessidade poderá ir adicionando mais atributos um sobre o outro. E mais, nada nos impede de adicionar uma descrição para nossa classe também. Observe que fazemos isso adicionando o atributo sobre a declaração da nossa classe.

Agora para que nosso mecanismo de reflexão (RTTI) possa extrair as informações dos atributos, teremos que fazer uma pequena alteração. A **Listagem 6** exibe esta alteração.

Na **Listagem 6** eu omiti a declaração das variáveis por não terem sofrido modificações. As linhas adicionadas estão em negrito, são elas: linhas 4 a 8 e 14 a 18. Essas linhas são responsáveis por retornar a lista de atributos que um tipo possa ter. Este tipo pode ser uma classe, um método, um propriedade, etc. No nosso caso primeiro retornamos os atributos da classe. Para cada atributo retornado verificamos se o mesmo é uma descrição, linha 6 e 16. Caso seja nós fazemos um *Casting* para acessar o conteúdo da propriedade *Texto* do nosso atributo. Feito isso adicionamos mais um nó em nossa *TreeView* com o conteúdo do atributo descrição. A **Figura 2** mostra a aplicação em execução já com o nosso recurso.

Listagem 6. Modificação do mecanismo de reflexão

```
1. begin
2.  Tipo := CT.GetType(Obj.ClassType);
3.  MainNode := TreeView1.Items.Add(nil,Obj.ClassName);

4.  for Att in Tipo.GetAttributes do
5.  begin
6.   if Att is Descricao then
7.    TreeView1.Items.AddChild(MainNode,Descricao(Att).Texto);
8.  end;

9.  TypeNode := TreeView1.Items.AddChild(MainNode,'Propriedades');
10.   for Prop in Tipo.GetProperties do
11.   begin
12.   AttNode := TreeView1.Items.AddChild(TypeNode,Prop.Name+'-'+
13.                                       Prop.GetValue(Obj).ToString);
14.    for Att in Prop.GetAttributes do
15.    begin
16.     if Att is Descricao then
17.      TreeView1.Items.AddChild(AttNode,Descricao(Att).Texto);
18.    end;
19.   end;

20.   TypeNode := TreeView1.Items.AddChild(MainNode,'Métodos');
21.   for Met in Tipo.GetMethods do
22.   begin
23.    TreeView1.Items.AddChild(TypeNode,Met.Name);
24.   end;
25. end;
```

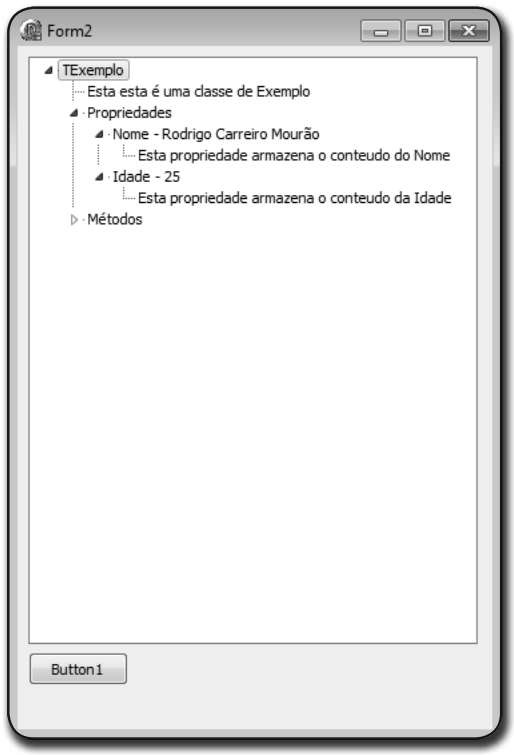


Figura 2. Aplicação exibindo os atributos

Conclusão

Talvez você esteja se perguntando onde utilizar esses novos recursos de reflexão. A resposta é: depende de sua necessidade e criatividade. Já pensou em criar um pequeno framework de validação de objetos, do tipo onde você possa informar se uma propriedade pode ser nula ou não, ou até mesmo verificar se uma string digitada está no padrão informado? Lembre-se, um TField ou um TClientDataset são classes, ou seja, mesmo seu sistema não sendo orientado a objetos você pode se beneficiar da reflexão e criar alguns atributos para seus DataSets, centralizando assim algo que poderia estar espalhado. Você poderia também criar como o que foi mostrado acima, um atributo conteria maiores detalhes de um determinado campo ou propriedade. Dessa forma ao colocar esse campo ou propriedade em algum formulário, você poderia exibir essas informações complementares, facilitando a vida do usuário. Outra excelente implementação prática do uso de RTTI é na criação de frameworks de persistência e mapeamento objeto / relacional. Essas são apenas algumas utilizações possíveis. Agora você pode praticar e se aprofundar no assunto e não se acanhe. Mantenha a mente aberta e verifique o que já foi feito com o uso de reflexão e custom attributes em outros ambientes como Java e .Net e talvez, até porque não migrar algo que existente neles que seja interessante, para o Delphi. ●

Links

Framework de persistência em Delphi: TiOPF
http://tiopf.sourceforge.net/

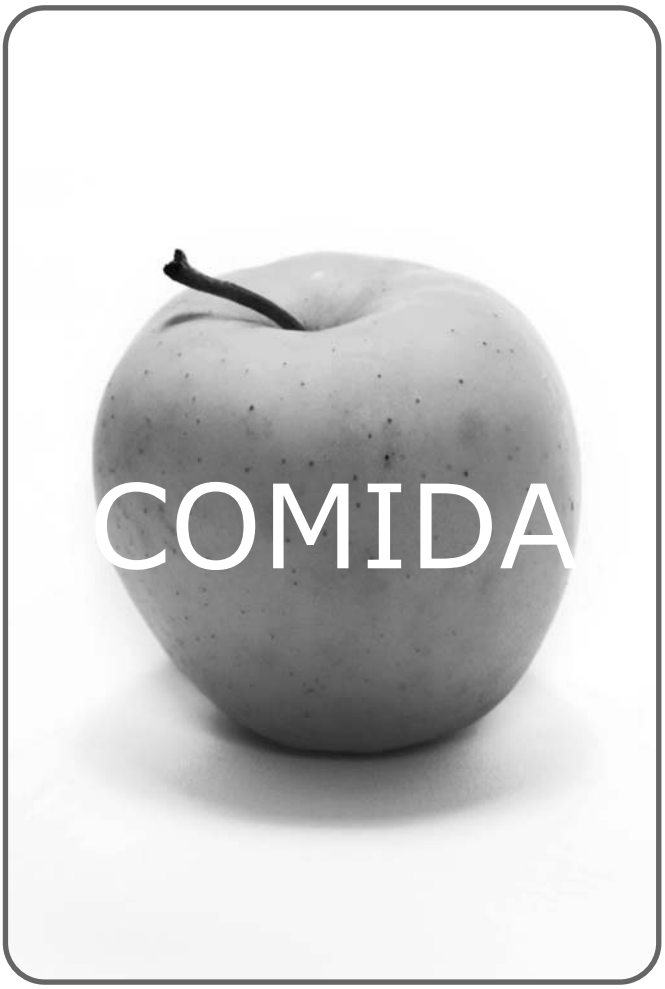
Framework de persistência em Delphi: InstantObjects
http://www.instantobjects.org/

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Existem coisas
que não conseguimos
ficar sem!

...só pra lembrar,
sua assinatura pode
estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à POO – Parte 4

Aprenda de uma forma simples a programar de forma correta

Do que trata o artigo
Este artigo aborda a orientação a objetos com o Delphi, usando uma metodologia simples, didática, de fácil aprendizado. Veremos na teoria, e também na prática, todos os conceitos, fundamentos e recursos oferecidos pelo Delphi para promover a POO.

Para que serve
A POO pode e deve ser aplicada de forma inteligente, ela serve para construir sistemas mais robustos, mais confiáveis, de fácil manutenção, que permitam maior reaproveitamento de código.

Em que situação o tema é útil
A POO é útil em qualquer sistema, seja ele Web, Desktop, Mobile, não importa o tipo de aplicação. Os conceitos aqui apresentados, em um exemplo simples, podem ser utilizados em aplicações reais, como é apresentado em um estudo de caso no final desta série.

Guinther Pauli
guinther.pauli@gmail.com
Blog: http://guintherpauli.blogspot.com
http://twitter.com/guintherpauli
Tem 22 anos de experiência na área de desenvolvimento e arquitetura de software. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Kylix, certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD. Editor da Revista ClubeDelphi e da Revista .NET Magazine Brasil, professor de Pós-Graduação. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas. Foi arquiteto e um dos desenvolvedores, voluntário, do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Resumo do DevMan

Neste artigo vamos usar o que o Delphi tem de melhor desde a sua primeira versão: componentes. Ou seja, com o IDE e a linguagem Delphi, temos todo o poder da OO aliado à velocidade RAD da ferramenta. Vamos hoje transformar nossas classes em componentes, examinando importantes conceitos base para promover a persistência, como RTTI.

Até aqui já aprendemos muito sobre *classes*, *herança*, *abstração*, *encapsulamento*, grandes pilares da orientação a objetos. E como se já não bastasse, colocamos um ingrediente especial na receita, o *polimorfismo*, o que considero ser a técnica mais poderosa da *OO*, a capacidade de comandar o presente deixando a implementação aberta para o futuro. Se adicionarmos *classes abstratas* e *interfaces*, então, a arquitetura fica ainda mais flexível.

O que nos interessa neste artigo é dar um passo além das classes. Classes são ótimas, mas se você quer aproveitar o melhor do *RAD* do *Delphi*, dos editores, usará componentes. Praticamente tudo o que fazemos hoje no *Delphi* é usando, ligando e configurando componentes. Antes de mais nada e para não deixar dúvidas, componentes são classes, que obrigatoriamente descendem de *TComponent* ou um dos seus vários descendentes.

TPersistent, classe base de *TComponent*, permite que objetos sejam *serializados*. *Serializar* um objeto significa persisti-lo em disco (leia-se suas propriedades) e posteriormente ler esses dados de volta para o objeto. Essa técnica é a chave para o uso de *frameworks* de persistência e mapeamento objeto / relacional, como *Hibernate*, *NHibernate*, *ADO.NET Entity Framework*, *ECO* etc. A *persistência* (mais exatamente serialização) de objetos também é usada para transmissão de dados via rede, por exemplo, em uma aplicação multicamadas. No *Delphi*, os objetos (ou melhor, componentes) são serializados em arquivos *.DFM*, graças a todo *TComponent* ser obrigatoriamente um *TPersistent*. Para um melhor entendimento da hierarquia do *core* da *VCL*, veja a **Figura 1.** Você não instancia nenhuma dessas classes diretamente, elas são base para classes mais específicas, como formulários, botões, componentes de acesso a dados etc.

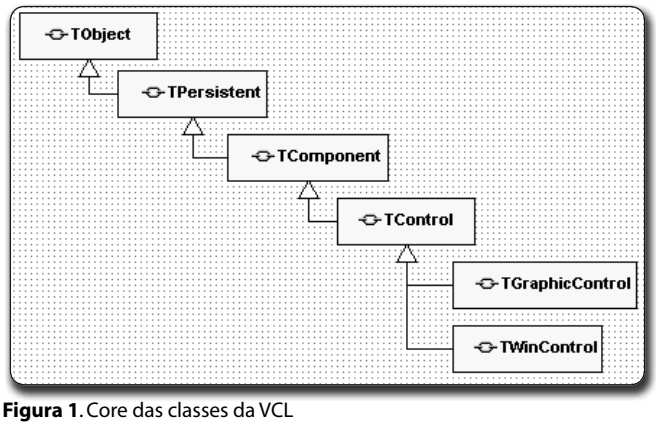


Figura 1. Core das classes da VCL

É *TPersistent* que ativa a *RTTI* (*Runtime Type Information*). *RTTI* é o grande segredo da persistência de objetos no *Delphi*. Quer ver? Abra os fontes de qualquer projeto seu em *Delphi*, segure a tecla *Control*, clique sobre a classe *TForm* que está no cabeçalho da declaração do seu *form* e vá clicando nas classes e subindo até chegar em *TPersistent*. Chegaremos ao seguinte:

```
{$M+}
TPersistent = class(TObject)
```

O que você vê acima da classe é uma diretiva de compilação, que ativa a *RTTI* para todos os descendentes de *TPersistent*. Convenhamos, "M" não diz nada para quem lê o código, obrigatoriamente o desenvolvedor vai ter que buscar no *Help* ou *Google* para descobrir o que é. Então, as diretivas também têm uma versão extensa. Por exemplo, o seguinte seria equivalente ao exemplo acima, bem mais intuitivo:

```
{$TYPEINFO ON}
TPersistent = class(TObject)
```

Nota do DevMan

Muitas das diretivas de compilação podem ser configuradas no Project>Options. Por exemplo, se você marcar a opção Complete Boolean Eval, na verdade vai estar adicionando a seguinte diretiva ao seu projeto:

{$B+}
Ou o equivalente:
{$BOOLEVAL ON}
Essa diretiva é relativa à análise de expressões booleanas, por exemplo:
if (1 = 1) or (3 <> 3) then ...
Veja que não há motivos para o compilador processar o segundo teste (3 <> 3), pois o primeiro teste (1 = 1), retorna true, e como o operador é um or, true com false vai ser sempre true (lembra das aulas de lógica dos tempos da faculdade?). Então, o padrão é {$B-}.
Essas configurações ficam gravadas no arquivo [NomeDoProjeto].cfg. Se você não quiser usar o arquivo .cfg, há uma técnica interessante que permite adicionar as diretivas atuais do projeto ao próprio fonte, nesse caso o local mais indicado é no próprio .dpr. Faça um teste, abra o .dpr e logo abaixo do program aperte Ctrl+O.

Graças à ativação da *RTTI* logo em *TPersistent*, todo *TComponent* pode ser serializado em *.dfm*. É por esse motivo que você configura as propriedades de um *ClientDataSet*, *Button*, *Memo*, *ListBox*, fecha o projeto, e quando reabre, tudo está lá configurado.

Dica do Editor

O melhor livro que já li que aborda RTTI, entre outras coisas, foi o Delphi in a Nutshell, de Ray Lischner. Ele está disponível em quase sua totalidade para leitura on-line no Google Books (veja sessão ***links***).

Em nosso projeto, *TCarro* e *TAviao* não são persistentes. Isto é, se você criar um ASTRA com capacidade 5 e quilometragem 5 mil quilômetros, quando o objeto for destruído e a aplicação fechada, já era. Então, vamos dar vida eterna aos nossos objetos, tornando-os persistentes, indo direto ao ponto, transformando classes em componentes.

Nota

A partir desta parte do artigo, nosso projeto vai ser migrado para Delphi 2010. Não se preocupe caso ainda esteja usando o Delphi 7 (dá para continuar o curso nele). No entanto, não abordarei ainda as novidades do RTTI que são inúmeras na versão 2010 (confira um artigo nessa edição sobre as novidades).

Criando um novo pacote e um novo componente

Componentes obrigatoriamente ficam dentro de pacotes. Um pacote é, basicamente, um conjunto de units que quase sempre definem componentes, embora você possa ter *units* que não contêm componentes. Um pacote permite, entre outras coisas, que uma aplicação fique "*modularizada*". Pacotes podem diminuir o tamanho do seu executável final e auxiliar na distribuição de atualizações. Um pacote, depois de compilado, é um arquivo *.bpl* (um tipo especial de *.dll*), e pode ser de 3 tipos: *RunTime*, *Design-Time* e *RunTime & Design-Time*. A extensão

do arquivo-fonte do projeto é *.dpk*. A título de curiosidade, o suporte a pacotes surgiu no *Delphi 3*.

O ponta pé inicial então é criar um pacote. Clique em *File>New>Package* e salve o pacote no mesmo diretório da aplicação que estamos construindo, para facilitar. Chame o pacote de "*pkPessoa*". Antes de adicionarmos nossas classes ao pacote, vamos criar um novo componente nele. Clique em *Component>New VCL Component*, no editor que se abre escolha a classe base para nosso novo componente, nesse caso, *TComponent* (**Figura 2**). Clique em *Next* e preencha os dados da tela como mostra a **Figura 3**. Repare: nome do componente será *TPessoa*, a paleta que ficará é *ClubeDelphi*, o nome da *unit* é *uPessoa.pas* e deve ficar no mesmo diretório da aplicação. Clique *Next*, na próxima tela escolha *Install to Existing Package*, localize nosso pacote recém criado, salve a *unit* e pronto. O editor do pacote deve agora estar como mostra a **Figura 4**.

Nota

Se você quiser compilar suas classes (.DCU) em um outro diretório, o recomendado, crie uma Lib e indique esse caminho no Tools>Options>Library Path.

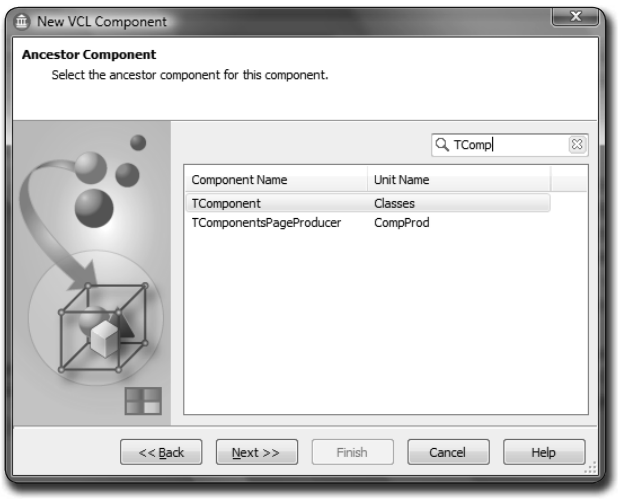


Figura 2. Criando um novo componente e escolhendo o ancestral

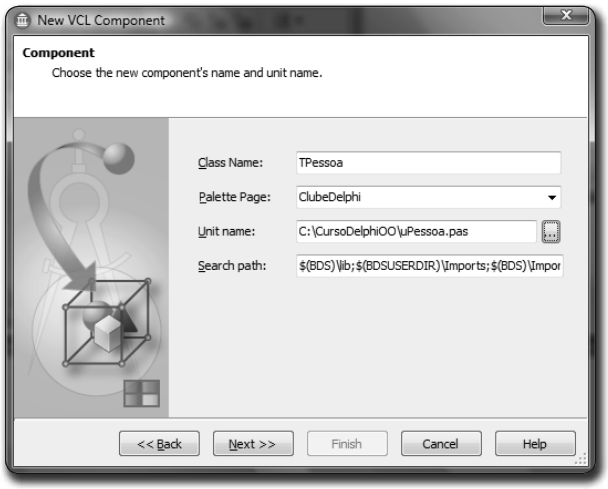


Figura 3. Classe se chamará TPessoa e ficará na unit uPessoa no diretório da aplicação

Nota do DevMan

Você já parou para pensar por que existe a classe TPersistent no meio de TObject e TComponent? Por que a implementação da persistência não foi feita em TComponent, se basicamente, somente componentes precisam ser persistidos? Ou seja, existe algum descendente de TPersistent que não seja TComponent? Sim. Isso é aplicado constantemente na VCL em componentes que têm atributos de tipo classes que também precisam ser persistidos. Um exemplo clássico:

TStrings = class(TPersistent)

Você não larga um TStrings na tela, pois ela descende diretamente de TPersistent, é por esse motivo que você digita um texto em um Memo (em design), fecha o projeto, e ao reabri-lo lá estão as linhas. A propriedade Lines não é um componente, mas precisa ser persistente.

Outro exemplo é a classe TPicture. Não colocamos essa classe no form, pois não é um componente, mas quando você coloca um Image e configura a propriedade Picture para mostrar um banner da empresa no topo do form, lá fica ela persistida no .dfm.

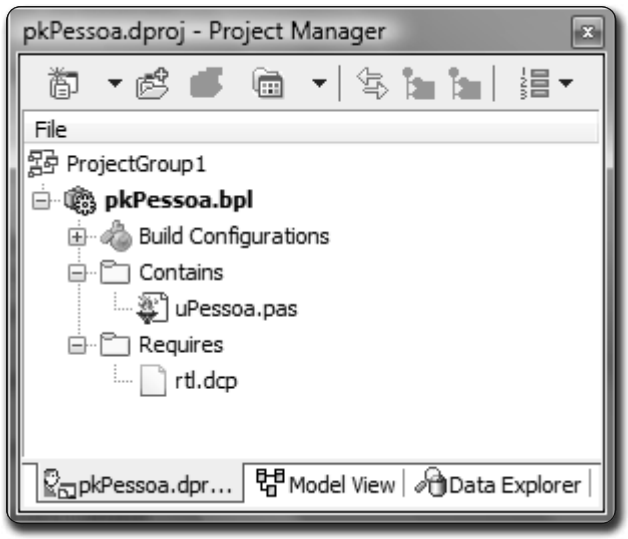


Figura 4. Editor do pacote

Vamos alterar a classe *TPessoa*, abrindo a *unit uPessoa.pas* e adicionando a ela a seguinte propriedade na sessão *published*:

```
property Nome: string;
```

Aperte *Shift+Ctrl+C* para que o *Delphi 2010* complemente a propriedade gerando o *Field / getter* e *setter* (estudamos isso no artigo da edição passada). Salve tudo, dê um *Build* no pacote, no editor dê um clique de direita no nome projeto e escolha *Install*. Pronto, a classe, ou seja, componente, está agora disponível na paleta de componentes em *ClubeDelphi*.

Vamos fazer um teste simples, inicie uma nova aplicação *VCL* e coloque no *form* nosso componente recém criado. Configure um nome para a pessoa, aperte *Alt+F12* e veja que isso foi persistido no *.dfm*. Salve tudo, reabra o projeto, e veja que nosso objeto é reconstruído com todas as suas propriedades, ou seja, persistência na prática (**Figura 5**). Repare que, como

nossas propriedades foram declaradas com o modificador *published*, elas aparecem no *Object Inspector*.

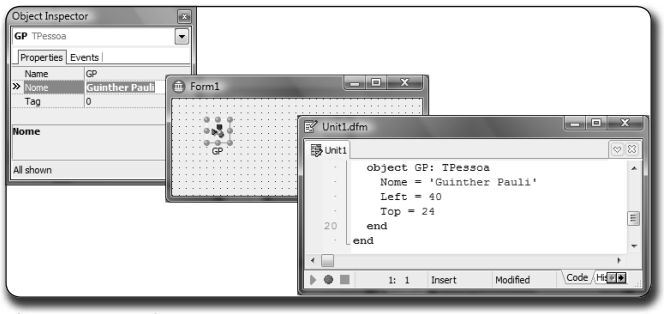


Figura 5. Testando componente e persistência

Transformando Classes em Componentes

O que faremos agora é transformar em componentes nossas três classes criadas nas partes anteriores deste curso (*TMeioTransporte, TCarro* e *TAviao*). Primeiro, reabra o projeto do pacote e com o item *Contains* selecionado, clique no botão *Add*. Adicione as *units uMeioTransporte, uCarro* e *uAviao*. Atenção, não coloque o form.

Abra a unit *uMeioTransporte* e defina *TComponent* como classe ancestral de *TMeioTransporte*. Adicione *Classes* à cláusula *uses*. Como agora nossa classe não descende mais de *TObject*, e sim *TComponent*, o *construtor* precisa ser modificado. Primeiro, ele agora deve receber um proprietário (*Owner*). Segundo, em *TComponent* ele é *virtual*, logo, precisamos usar *override* (lembra do *polimorfismo*? Olha ele de novo aqui, auxiliando os construtores). O código completo da classe pode ser visto na **Listagem 1**. Nas classes *TAviao* e *TCarro* faça a mesma coisa, declare *classes* no *uses* e modifique o construtor. Lembre-se de que o construtor deve ser modificado na *interface* e *implementation*.

Listagem 1. TMeioTransporte agora é componente

```
unit uMeioTransporte;

interface

uses Classes;

type
  TMeioTransporte = class(TComponent)
  private
    FCapacidade : integer;
    FDescricao : string;
  protected
    procedure Ligar(); virtual; abstract;
  public
    procedure Mover(); virtual;
  published
    property Capacidade: integer read FCapacidade write FCapacidade;
    property Descricao: string read FDescricao write FDescricao;
  end;

implementation

{ TMeioTransporte }
uses
  Dialogs;

procedure TMeioTransporte.Mover();
begin
  Ligar();
end;

end.
```

Por tabela, ou melhor dizendo na linguagem *OO*, por *herança*, agora *TCarro* e *TAviao* são componentes também. E já temos um pequeno *framework*. Agora precisamos indicar ao *Delphi 2010* para registrar os componentes *TCarro* e *TAviao* no *IDE*. Vá até a unit *uCarro* e declare o seguinte procedimento imediatamente **antes** da cláusula *implementation*:

```
procedure Register;
```

Implemente esse procedimento na seção *implementation* da *unit*, logo após a cláusula *uses*:

```
procedure Register;
begin
  RegisterComponents('ClubeDelphi',[TCarro]);
end;
```

RegisterComponents registra o(s) componente(s) no *IDE* na paleta indicada no primeiro parâmetro. O segundo parâmetro é um *array* de *TComponentClass* (referência de classe). Faça o mesmo para unit *uAviao*. Recompile o pacote e instale os novos componentes. Veja os componentes instalados na **Figura 6**.

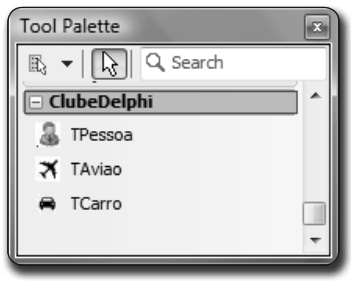


Figura 6. Componentes instalados na Tool Palette

Nota

Veja que não registramos TMeioTransporte, pois a classe não é usada diretamente, é uma classe abstrata que serve de base para as classes mais específicas. Pelo mesmo motivo, é que você vai encontrar na paleta um ClientDataSet, SqlDataSet, Table, mas nunca um TDataSet.

Polimorfismo e abstração em componentes

Agora imagine que um objeto *Pessoa* precise fazer uso de um *Carro* e de um *Aviao*. Poderíamos simplesmente declarar duas propriedades na classe *TPessoa* (atenção, não altere seu código, apenas observe):

```
property Carro : TCarro read FCarro write FCarro;
property Aviao : TAviao read FAviao write FAviao;
```

Nota do DevMan

Curiosidade

A linguagem Delphi, como sabemos, não diferencia maiúsculas e minúsculas (case-insensitive). A única exceção é o procedimento Register, que deve ter "R" maiúsculo.

Nota do DevMan

Como colocar ícones nos componentes

Abra o Image Editor do Delphi. Clique em File>New>Component Resource File (.dcr). Salve o arquivo no mesmo diretório do pacote, como nome pkIcones.dcr. Clique de direita em Contents e escolha New>Bitmap. Defina as dimensões 24x24 e 256 cores. Renomeie Bitmap1 para o mesmo nome da classe, tudo em maiúsculo, neste caso "TPESSOA". Dê um duplo clique nesse item e desenhe ou cole o ícone no editor que aparece. Repita o mesmo procedimento para as demais classes. Salve o arquivo. Volte ao pacote no Delphi 2010 e clique em Project>View Source. Logo após a declaração do pacote inclua o arquivo .dcr como abaixo e recompile.

```
package pkPacote;
{$R pkIcones.dcr}
```

Pronto, uma pessoa poderá agora utilizar carros e aviões. Mas temos um problema. Se um dia for criado um novo meio de transporte, como *TBike*, nosso componente *TPessoa* precisaria de uma nova propriedade. E se novos meios de transporte forem surgindo, serão necessárias novas alterações em *TPessoa*. E pior, nós nunca vamos saber que tipos de meio de transporte poderão existir no futuro. Quem sabe um *TTeleTransporte?* Na *OO*, temos que programar pensando em **baixo acoplamento,** ou seja, depender o mínimo possível de outras classes, cuidando para não sofrer com suas mudanças ou novas implementações. Aí entra em cena outra técnica, *abstração*. E conseguimos programar para *abstração* usando classes bases *abstratas*, mais genéricas, ou *interfaces*. Devemos sempre programar pensando nessas classes bases, no padrão, na *interface*.

Já imaginou a tecla *Enter* do seu notebook estragar e você ter que trocar a tela, *CPU*, *mouse*, *pen-drive*, *WebCam*, *HD*, *driver DVD*, *adaptador de rede*, *wireless* etc. E se o pneu do seu carro furar e ficar sem concerto, e por tabela você ter que trocar todo o sistema elétrico? (notou que um não pode depender do outro?) Não é assim que funciona na vida real, não é assim que funciona na orientação a objetos, não é assim que funciona no *Delphi*.

Lembra daquela analogia que fiz nas partes anteriores do artigo? Você não tem um furo na lateral do micro para cada tipo de dispositivo, você usa um padrão, *USB*, e nele pluga qualquer dispositivo. E no *Delphi*? Mesma coisa. Um *QuickReport* aponta para um *TDataSet*, um tipo mais genérico, o que torna a arquitetura plugável, extensível. Ou seja, hoje eu conecto o *QR* a qualquer tipo descendente de *TDataSet*. O mesmo acontece com o *TDataSource*, ele esconde (abstrai) dos *DataControls* qual o tipo específico de *TDataSet* será usado, ou seja, ele joga um padrão.

Em nosso exemplo, vamos fazer exatamente a mesma coisa que a *VCL* faz e que a *OO* manda. Vamos criar uma única propriedade que possa apontar para qualquer meio de transporte. Então declare a seguinte propriedade na sessão *published* de *TPessoa*:

```
property Transporte : TMeioTransporte read
FTransporte write FTransporte;
```

Aperte *Shift+Ctrl+C* para gerar a declaração do campo privado. Declare *uMeioTransporte* na cláusula *uses*. Recompile o pacote. Feito, essa é a nossa *"entrada USB"*.

Criando uma aplicação para testar os componentes

Inicie uma nova aplicação *VCL* e salve os arquivos do projeto. Coloque no formulário principal um *Button* com o *Caption "Mover"*. Coloque um *Carro* e uma *Pessoa*. Defina as propriedades do objeto *Carro*. Aponte a propriedade *Transporte* da *Pessoa* para o objeto *Carro* (veja **Figura 7**). O código *.dfm* do *form* é mostrado na **Listagem 2.**

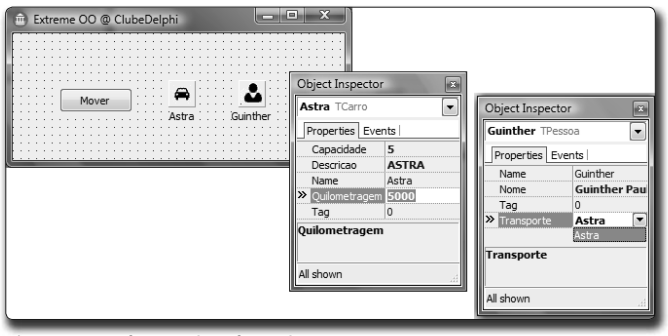


Figura 7. Configurando o formulário

Listagem 2. Código .dfm do formulário

```
object Form1: TForm1
  Caption = 'Extreme OO @ ClubeDelphi'
  ...
  object Button1: TButton
    Caption = 'Mover'
    OnClick = Button1Click
    ...
  end
  object Astra: TCarro
    Capacidade = 5
    Descricao = 'ASTRA'
    Quilometragem = 5000
    ...
  end
  object Guinther: TPessoa
    Nome = 'Guinther Pauli'
    Transporte = Astra ←repare o componente apontando para o outro
    ...
  end
end
```

No evento OnClick do botão Mover escreva:

```
Guinther.Transporte.Mover();
```

Execute a aplicação e clique no botão, veja o resultado na **Figura 8**.

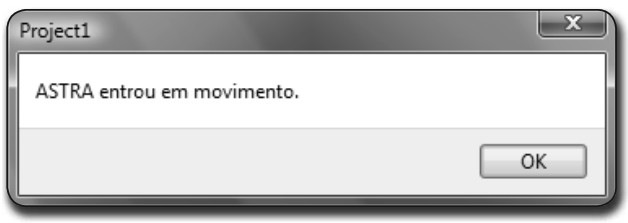


Figura 8. Polimorfismo via componentes abstraídos

Agora coloque um componente *Avião* no formulário, e configure suas propriedades. Aponte a propriedade *Transporte* da *Pessoa* para o objeto *Aviao*. Clique no botão e veja que a pessoa pode também colocar um avião em movimento, graças ao *polimorfismo* (**Figura 9)**.

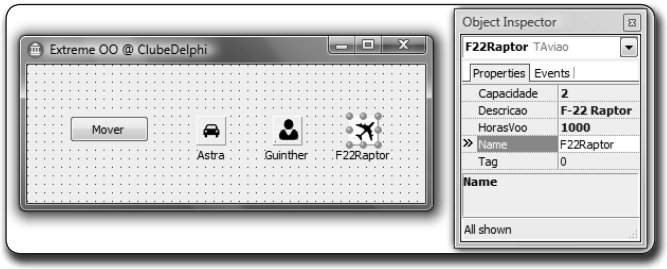


Figura 9. Pessoa pode mover um avião

O que isso tudo significa? Que conclusões podemos tirar?

- Nossos objetos agora são persistentes. Você pode fechar o projeto, reabrir, que as propriedades estarão lá, configuradas, graças à *RTTI* e persistência;

- A *Pessoa* possui um atributo chamado *Transporte*, que não é de um tipo específico, mas de um tipo genérico, *TMeioTransporte*. Isso abstrai da pessoa detalhes específicos do tipo do meio de transporte que ela vai operar;

- A abstração torna nossa arquitetura plugável e extensível. Se criarmos um *TBike*, ótimo, basta a pessoa apontar para o novo objeto que já terá condições de colocá-lo em movimento, graças ao polimorfismo. Estamos programando no presente prontos para o futuro;

- E finalmente, é exatamente assim que a *VCL* trabalha. Compare as duas linhas abaixo, a do nosso exemplo, e de outro exemplo que seria usando componentes de acesso a dados do *Delphi*, alguma semelhança?

```
Guinther.Transporte.Mover();
DataSource1.DataSet.Post();
```

Mover é polimórfico, então não importa para que meio de transporte a pessoa esteja apontando, o método correto será invocado. No *DataSource*, o *TDataSet* pode apontar para um *TClientDataSet*, *TTable*, não importa, o método polimórfico será chamado de acordo com a classe instanciada (e não a declarada, o que caracterizaria um método estático).

Mais uma coisinha que você precisa saber. Quer ver o *IDE* dar pau, travar tudo e obter inúmeros *AV's* (*Access Violation's*)? Apague o componente *Aviao*, ou aquele que esteja sendo apontando pela propriedade *Transporte* da *Pessoa*. O que vai acontecer? A *Pessoa* não vai ficar sabendo que o objeto que ela estava apontando foi excluído, logo, vai apontar para algo que não existe mais. Para isso, precisamos fazer que a propriedade *Transporte* seja zerada (*nil*) no momento em que o componente de transporte associado for excluído. Fazemos isso através do *Notification*. De volta ao pacote, declare o método na seção *protected* da classe *TPessoa* e implemente-o como na **Listagem 3.** Não se esqueça de dar um *Build* no pacote.

Listagem 3. Notification

```
procedure Notification(AComponent: TComponent;
  Operation: TOperation); override;
…
procedure TPessoa.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited;
  if (Operation=opRemove) and
    (AComponent=Transporte) then
    Transporte := nil;
end;
```

Eventos

Muito bem, nossos componentes contêm propriedades, inclusive as herdadas de *TComponent* (*Name* e *Tag*). Porém, se você clicar em *Events* no *Object Inspector*, verá que nada existe. Muitos consideram o uso de eventos algo que vai de encontro à orientação a objetos. Mas em um ambiente *RAD* como o *Delphi*, eventos jogam a seu favor, e muito. Vamos ver como poderíamos implementar um evento em nossa hierarquia. Vamos criar um evento chamado *BeforeLigar*. Então, o desenvolvedor que for usar nosso componente, poderá usar esse evento que será disparado antes do transporte entrar em movimento. É mesma coisa do *TDataSet*. Você ou algum controle de tela chama *Post*, e o *TDataSet* dispara o *BeforePost* para você programar por exemplo alguma validação.

Declare o seguinte na seção *published* da classe *TMeioTransporte*:

```
property BeforeLigar : TNotifyEvent;
```

Aperte *Shift+Ctrl+C* e altere o método *Mover* da classe como mostra a **Listagem 4.**

Recompile o pacote. Volte ao exemplo anterior. Selecione o componente *Carro* e veja o novo evento no *Object Inspector* **(Figura 10).** Adicione o código da **Listagem 5** ao manipulador do evento *BeforeLigar*.

Listagem 4. Chamando um evento

```
procedure TMeioTransporte.Mover();
begin
  if Assigned(FBeforeLigar) then
     BeforeLigar(self);
  Ligar();
end;
```

Listagem 5. Usando o evento

```
procedure TFrmExemplo.Carro1BeforeLigar(Sender: TObject);
begin
  raise EAbort.create('Isso cancelará a chamada aos métodos
Ligar e Mover');
end;
```

Figura 10. Eventos

Selecione o componente *Aviao* e aponte seu evento *BeforeLigar* para o mesmo método. Dessa forma, ambos os componentes usam o mesmo manipulador. O parâmetro *Sender* dirá qual dos componentes está chamando o evento.

Observe a exceção silenciosa que foi levantada caso o usuário deseje cancelar a chamada. O *raise* fará com que a sequência de chamadas (*Call Stack*) seja quebrada e todos os métodos cancelados. Sim, eventos também servem para isso!

Se você observar bem, o uso do evento fará com que o usuário do nosso *framework*, (e não o desenvolvedor dele), tenha a possibilidade de *"injetar"* código exatamente em um ponto específico da implementação da classe, usando tipos definidos de eventos. No nosso caso, o código é chamado antes do *Ligar*, para dar a oportunidade do usuário realizar algum procedimento antes do meio de transporte ser colocado efetivamente em movimento.

Conclusão

Simples, rápido, produtivo, elegante e extremamente poderoso. *OO* + Componentes + *RAD*, mistura ideal para criar projetos complexos, tornar o código reaproveitável, modularizado, ao mesmo tempo que fazemos uso de boas práticas. Desde o começo deste curso, aprendemos muitas coisas. Tudo bem que foram com carrinhos e aviões, mas veja do ponto de vista do *design*: é exatamente assim que a *VCL* foi modelada, é exatamente assim que a vida real funciona, é assim que você deve desenvolver. Abraço e até a edição 112. ●

Links

Livro Delphi in a Nutshell no Google Books (e outros bons livros)
http://guintherpauli.blogspot.com/2009/08/livros-de-delphi-no-google-books.html

GExperts
http://www.gexperts.org/

Blog do autor
http://guintherpauli.blogspot.com
http://twitter.com/guintherpauli

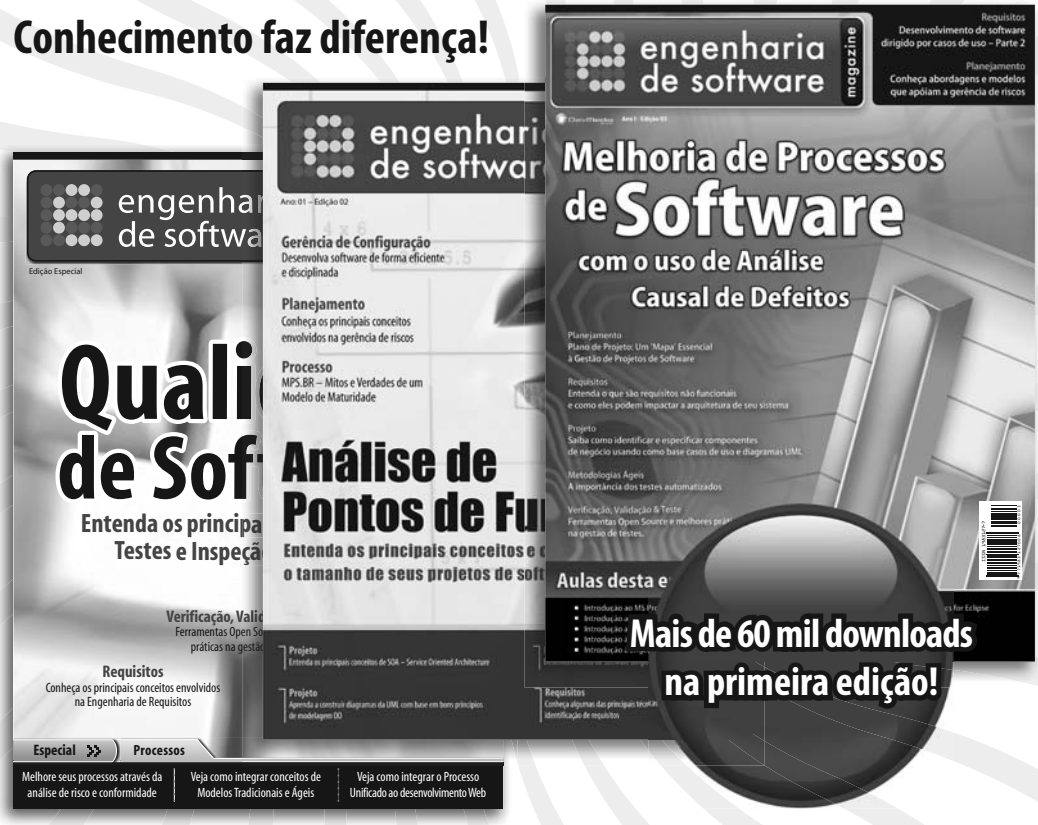
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conhecimento faz diferença!



Faça já sua assinatura digital! | *www.devmedia.com.br/es*

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia GROUP

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Smarty Template

Separe a interface gráfica da lógica de programação

Do que trata o artigo
O artigo que será apresentado tem como objetivo mostrar como podemos trabalhar com a biblioteca Smarty Template, incluindo instalação, considerações iniciais, primeiro exemplo, arquivo de configuração, banco de dado e exemplo com banco de dados. No final do mesmo será apresentado um exemplo com a ferramenta Delphi For PHP.

Para que serve
Mostrar ao leitor a biblioteca Smarty Template, que resulta em produtividade na aplicação, fácil manutenção e separação da interface gráfica com a lógica de programação.

Em que situação o tema é útil
Trabalhar com a biblioteca Smarty Template pode nos trazer muitas vantagens, entre elas citamos você ficar responsável por toda parte lógica da aplicação Web, e o designer pela interface gráfica.

Resumo do DevMan

A biblioteca Smarty é um sistema de templates para PHP. Mais especificamente, ela fornece uma maneira fácil de separar a lógica de programação de sua interface gráfica. Isto é mais bem descrito em uma situação onde o programador da aplicação e o designer do layout executam diferentes funções, ou na maioria dos casos não são a mesma pessoa. A mesma também está disponível no Delphi for PHP.

Rafael Silva
rafael@rlsystem.com.br
Trabalha com análise e desenvolvimento de sistemas. Atua na área de desenvolvimento desktop utilizando Delphi, C# e foco na Web com PHP, ASP.NET, CSS, AJAX, sempre utilizando os padrões da Web 2.0 em seus projetos. Utiliza banco de dados MySQL, SQL Server, PostgreSQL e Firebird.

Hoje em dia a complexidade de criar *Web Sites* aumentou bastante, principalmente quando os tipos de aplicações desenvolvidas são intranet, portais etc. Com isso a manutenção a ser realizada aumentou e dificultou bastante. Quem utiliza *PHP* sabe que ele é a melhor solução *open-source* para desenvolvimento Web. Mas enfrentamos um problema onde o código de programação fica acoplado junto com as *tags HTML*. Outro problema que

temos é quando vamos trabalhar em equipe, com *Web Designers*, onde desenvolvemos nossa aplicação e ele fica responsável por aplicar o *layout* no site. Porém geralmente *Designers* não têm conhecimento de programação e o que acaba acontecendo são erros na aplicação ou mesmo o projeto tendo que ser devolvido para o programador realizar os ajustes.

Com certeza você já desenvolveu um site para seu cliente e ele logo após algum tempo solicitou um novo *layout*. Isso é muito comum. Para ajudar podemos usar alguns recursos como o famoso *include*, utilizado para adicionar determinado arquivo / conteúdo na página. Assim toda sua estrutura é concedida a diversas páginas e a manutenção no site fica facilitada. Mesmo com essa solução nossos arquivos também conteriam códigos *php* acoplados com *HTML*.

Neste artigo será apresentada a biblioteca *Smarty Template*, que **separa a interface gráfica da lógica de programação**. A biblioteca *Smarty Template* é também composta por alguns pontos importantes, como a utilização de *tags* nos *templates*, compilação das páginas (reduzindo o tempo da mesma ser exibida), segurança, arquivos de configuração, fácil manutenção da aplicação etc. Nos tópicos a seguir serão citadas algumas características que tornam a biblioteca *Smarty Template* como a melhor e mais robusta biblioteca de *templates* para *PHP*.

Instalação

Antes de iniciarmos nossos estudos com a *Smarty Template*, devemos instalar a biblioteca. Para isso você deve acessar o site da *Smarty* e realizar o *download* (veja o endereço na seção *Links*). Logo após é necessária a descompactação dos arquivos em um diretório a sua escolha. Lembre-se que os exemplos criados serão todos executados em um servidor local.

Veja na **Figura 1** a lista de diretórios necessários para a utilização da biblioteca**.** A seguir você encontra a lista de todos os diretórios com uma breve descrição de funcionamento. Não esqueça que o diretório *Libs* não deve ser criado e sim copiado do *download* que foi realizado anteriormente.

Veja a seguir uma descrição das pastas:

/configs = Arquivos de configurações;

/ libs = Biblioteca que contém todas as classes da *Smarty Template*;

/templates = Diretório que contém os *templates* do projeto (interface gráfica da aplicação);

/templates_c = Diretório que contém o cache dos *templates*. É necessária permissão de escrita no diretório (*CHMOD* 777).

Considerações inicias

Antes de iniciarmos a criação de nossos primeiros exemplos, temos que estar ciente de como funciona a Smarty Template. Para isso citarei algumas informações essenciais para que você esteja apto e capacitado a utilizar a biblioteca Smarty Template.

Todos os templates criados devem receber a extensão tpl (por exemplo, home.tpl, noticias.tpl, artigos.tpl, clientes.tpl). Como você pode observar, esses arquivos são nossos arquivos de templates, mais especificamente são a interface gráfica da aplicação Web. Tudo que estiver relacionado com imagens, css, HTML e outros deve estar contido nesses arquivos. Se os arquivos de template ficam em arquivos com extensão tpl, onde devemos concentrar a lógica de programação? Simplesmente em arquivos php, que contêm toda a parte lógica da aplicação, sem nenhuma tag HTML.

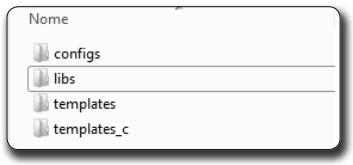


Figura 1. Diretórios necessários que devem ser criados para trabalhar com a biblioteca Smarty Template

Primeiro exemplo

Vamos começar então a criar nosso primeiro exemplo com a biblioteca *Smarty Template*. Como é de costume, vamos criar um simples arquivo, onde teremos um exemplo "*Hello World*" (**Listagem 1**). Observe primeiramente que devemos chamar a biblioteca *Smarty* Para isso acontecer é necessário invocar a mesma com a função *include* (linha 2). Com isso temos acesso às propriedades e métodos da classe. É então criada uma instância da classe *Smarty*, que está sendo chamada de *$smarty* (linha 3). Também definimos algumas propriedades que são muito importantes para o funcionamento. A primeira propriedade é a *template_dir*, que tem por finalidade indicar onde estão localizados nossos arquivos de *templates*, no caso os arquivos com extensão *tpl* (linha 4). Mais abaixo temos o *compile_dir*, linha 5, que contém os *templates* compilados. E por último o *config_dir*, que contém os arquivos de configuração, linha 6 (não se preocupe, falaremos um pouco mais sobre arquivos de configuração adiante).

Veja que o método *assign* recebe dois parâmetros. O primeiro é o nome da variável que estamos criando, e o segundo é o valor dessa variável. Com esses parâmetros criados temos a possibilidade de exibir os valores em nossos *templates*. Estamos criando uma variável *msg* e definindo um valor *string Hello World*, linha 7. Por último temos o método *display*, linha 8, que recebe como parâmetro o nome do nosso *template* (*hello.tpl*), Você deve salvar esse arquivo como *hello.php* no diretório *root* da sua aplicação. Logo após é necessário criar um arquivo com a extensão *tpl* e salvar o mesmo no diretório *templates* com o mesmo nome que foi definido no método *display*, no nosso caso *hello.tpl*.

Nota

Você pode realizar o download da biblioteca Smarty Template no site oficial. Caso seja de seu interesse realizar a instalação de um servidor local em sua máquina, recomendo o Wamp Server (veja sessão Links). Ele é um pacote de componentes que instala em sua máquina o Apache, PHP e Mysql, em alguns poucos cliques.

Nota

Chmod é um comando Unix/Linux usado para alterar permissões de arquivos e diretórios (ou pastas). Sua sintaxe é composta por: chmod [permissões] arquivo

Vamos agora criar o nosso *template* que exibirá o conteúdo da variável *msg*. Você pode utilizar o bloco de notas ou o editor *HTML* que achar melhor. A **Listagem 2** exibe o conteúdo que deve estar contido em nosso arquivo *hello.tpl*. Observe a presença da *tag {$msg}*, ela será substituída pelo valor enviado pelo *PHP* através do método *assign* da *Smarty Template*.

Listagem 1. Exemplo Hello World

```
1. <?php
2. include "smarty/libs/Smarty.class.php";
3. $smarty = new Smarty();
4. $smarty->template_dir = "smarty/templates";
5. $smarty->compile_dir = "smarty/templates_c";
6. $smarty->config_dir = "smarty/configs";
7. $smarty->assign("msg", "Hello World!");
8. $smarty->display("hello.tpl");
9. ?>
```

Listagem 2. hello.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo Hello World</title>
</head>

<body>
Mensagem: {$msg}
</body>
</html>
```

Arquivo de configurações

Antes de abordarmos exemplos mais complexos, vou comentar um pouco sobre arquivos de configurações. Eles são muito úteis para *designers*, que podem gerenciar configurações a partir de um modelo global de variáveis. O exemplo que citarei é também informado no site oficial da *Smarty Template*.

O exemplo será um modelo de cores. Vamos imaginar que temos uma grande aplicação e temos como tarefa alterar suas cores de layout. Precisaríamos modificar cada arquivo da aplicação para atingirmos esse objetivo. Utilizando um arquivo de configuração seria necessário somente alterar um único arquivo, sem a necessidade de ir a cada um e modificar seu conteúdo.

Para esse nosso exemplo vamos criar um arquivo de configuração chamado *options.conf*, que deve ser salvo no diretório *configs*. Ele conterá algumas informações, como título do site, cor de fundo, tamanho da borda da tabela e endereço da empresa (**Listagem 3**).

Para executarmos o exemplo primeiramente devemos criar o arquivo *PHP* e *tpl* com os seguintes nomes: *exemplo2.php* e *exemplo2.tpl* (**Listagem 4**). Como comentado anteriormente, ambos devem estar em seus respectivos diretórios. Observe que em nosso *template* primeiramente carregamos o arquivo de configuração utilizando o *config_load* indicando no atributo *file* o nome do arquivo que queremos carregar.

Com isso ele é carregado com nosso *template* e de acordo com nossas regras setamos os valores retornados. Para utilizar estes valores no nosso *template* simplesmente utilizamos *{# nome da variável #}* (veja na **Figura 2** o resultado).

A seguir criaremos um exemplo simples com a biblioteca *Smarty Template*, mas utilizando banco de dados.

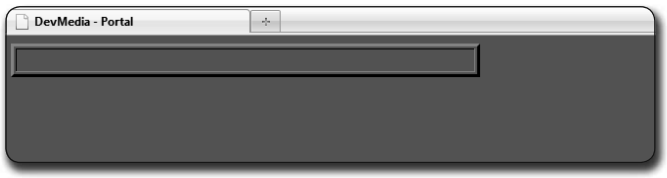


Figura 2. Exibição do arquivo Exemplo2.tpl

Listagem 3. Arquivo de configuração (options.conf)

```
PaginaTitulo = "DevMedia - Portal"
tabelaBorda = "3"
fundo = "#006699"
endereco = "rua de endereço x, número y"
```

Listagem 4. Exemplo 2 – Trabalhando com arquivo de configuração

```
exemplo2.php

<?php
include "smarty/libs/Smarty.class.php";
$smarty = new Smarty();
$smarty->template_dir = "smarty/templates";
$smarty->compile_dir = "smarty/templates_c";
$smarty->config_dir = "smarty/configs";
$smarty->display("exemplo2.tpl");
?>

exemplo2.tpl

{config_load file = 'options.conf'}
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>{#PaginaTitulo#}</title>
</head>

<body bgcolor="{#fundo#}">
<table width="480" border="{$smarty.config.tabelaBorda}">
<tr>
<td> </td>
</tr>
</table>
</body>
</html>
```

Extensão MySQLI

Antes de começarmos a utilizar a *Mysqli*, é necessário que a mesma esteja disponível em nosso servidor Web ou local. Hoje em dia por padrão ela já vem disponível em quase todos servidores de hospedagem que utilizam a versão 5.1.x do *PHP*. De qualquer forma é de extrema importância que você contate seu provedor e verifique se a extensão está instalada.

Aqui abordaremos a instalação da *Mysqli* em um servidor *Windows*, que utiliza um pacote de instalação denominado *Wamp Server*. É claro que não é uma exigência do mesmo, o requisito mínimo é você ter em sua máquina o *Apache*, *PHP* 5 e *MySQL*. Todas as configurações de funcionamento do *PHP*, e isso inclui principalmente extensões que ele deve carregar na sua inicialização, se encontram no arquivo de configurações *PHP.ini*.

É nesse arquivo que devemos indicar que o mesmo carregue a extensão *Mysqli*, se você estiver utilizando o pacote de instalação *Wamp Server*. O mesmo se encontra no diretório *C:\wamp\bin\apache\Apache2.2.11\bin\ PHP.ini* (ou *C:\Windows\PHP.ini*).

Abra o mesmo e procure a linha *extension=php_mysqli.dll*, certamente você encontrará um ponto e vírgula(;) no início da linha. Isso indica que o mesmo é um comentário, como consequência disso ele não será carregado. Retire-o caso haja. Com a linha alterada, você já pode salvar as alterações e reiniciar o *Wamp Server* (ou o *Apache*).

Criando a tabela

Vamos dar início à criação de um exemplo simples utilizando *Smarty Template*, com banco de dados *Mysql*. Antes de iniciarmos a codificação, devemos criar o banco de dados. O mesmo se chamará *cadastro* e conterá apenas uma tabela, no caso *Clientes*. Na **Listagem 5** você pode visualizar o *script SQL* para a criação da tabela *Clientes*, lembrando que você já deve ter criado o banco de dados.

Exemplo com banco de dados

A ideia do nosso exemplo é apenas exibir uma lista de clientes do banco de dados, tudo isso em conjunto com a biblioteca *Smarty Template*. Crie um arquivo *clientes.php* e um arquivo *clientes.tpl*, nos diretório *root* e *templates*, respectivamente (**Listagem 6**). No arquivo *PHP* instanciamos a classe *Smarty* e definimos valores nas suas propriedades (como visto nos primeiros tópicos). Logo abaixo é criada outra instância, mas da classe *MySQLI*, linha 7, que contém quatro parâmetros: endereço do servidor, usuário, senha e banco. Em seguida é executada uma consulta, linha 8, que a partir de um *ResultSet* é gerado um loop que segue armazenando os valores em um *array* (linhas 9 a 15). No arquivo *clientes.tpl* temos uma tabela *HTML*, que contém uma *section* para realizar um *loop*. Na **Figura 3** você visualiza o resultado da mesma.

ID	NOME	EMAIL	CIDADE
1	Rafael Silva	rafael@rlsystem.com.br	Guaiba
2	Rasmus Lerdorf	email@provedor.com	Groenlândia
3	Paulo João	paulo@terra.com.br	SP
4	Mário César	mario@uol.com.br	RJ
5	Adamastor Filho	adama@bol.com.br	Porto Alegre
6	Luiza Souza	luiza@test.com.br	Santos

Figura 3. Resultado do exemplo com banco de dados

Delphi for PHP

Depois de passados os conceitos importantes da biblioteca *Smarty Template* e também exemplos com PHP puro, será apresentado um exemplo utilizando a ferramenta *Delphi for PHP*. O *D4PHP* possui uma série de componentes que aumentam nossa produtividade de desenvolvimento e entre eles está a *Smarty template*. A ideia do exemplo que criaremos é um simples sistema de *login*. Não terá acesso a banco de dados, até porque não é o foco do artigo. Você que já trabalha com a ferramenta terá grande facilidade de agregar em seus projetos boas práticas, que é o foco do artigo, separando sua interface gráfica da lógica de programação.

Partindo do ponto que você já possua a versão 2.0 do *D4PHP*, vamos criar nosso projeto. Com o *D4PHP* aberto, clique em *File>New>Application*. Logo após, com o projeto já criado, clique em *File> Save as* e escolha um diretório de sua preferência, alterando o nome da *unit* que foi criada com o projeto para *index.php*.

Insira um componente *Panel* e altere sua propriedade *name* para *PDados*. Em seguida dentro do *Panel* acrescente dois *Label's* e altere suas propriedades *Caption* para *Login* e *Senha*, e *fonte-size* para 14. Logo após adicione dois *Edit's* definindo

Listagem 5. Script SQL

```
CREATE TABLE `Cadastro`.`Clientes` (
`ID` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`NOME` VARCHAR( 50 ) NOT NULL ,
`EMAIL` VARCHAR( 50 ) NOT NULL ,
`CIDADE` VARCHAR( 30 ) NOT NULL
) ENGINE = MYISAM
```

Listagem 6. Exemplo com banco de dados

```
Clientes.tpl

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1" />
<title>Untitled Document</title>
</head>

<body>
<table width="566" border="1">
  <tr>
    <td><div align="center">ID</div></td>
    <td><div align="center">NOME</div></td>
    <td><div align="center">EMAIL</div></td>
    <td><div align="center">CIDADE</div></td>
  </tr>
  {section name=i loop=$sID}
  <tr>
    <td>{$sID[i]}</td>
    <td>{$sNOME[i]}</td>
    <td>{$sEMAIL[i]}</td>
    <td>{$sCIDADE[i]}</td>
  </tr>
  {/section}
</table>
</body>
</html>

Clientes.php

1.  <?php
2.  include "smarty/libs/Smarty.class.php";
3.  $smarty = new Smarty();
4.  $smarty->template_dir = "smarty/templates";
5.  $smarty->compile_dir = "smarty/templates_c";
6.  $smarty->config_dir = "smarty/configs";
7.  $con = new mysqli('localhost', 'root', '123', 'cadastro');
8.  $query = $con->query("SELECT * FROM clientes");
9.  while($result = $query->fetch_array())
10.  {
11.  $id[]      = $result["ID"];
12.  $nome[]    = $result["NOME"];
13.  $email[] = $result["EMAIL"];
14.  $cidade[]  = $result["CIDADE"];
15.  }
16.  $smarty->assign("sID", $id);
17.  $smarty->assign("sNOME", $nome);
18.  $smarty->assign("sEMAIL", $email);
19.  $smarty->assign("sCIDADE", $cidade);
20.  $smarty->display("clientes.tpl");
21.  ?>
```

suas propriedades *Name* para *txtLogin* e *txtSenha*. Para finalizar, acrescente um *Button* que irá submeter essas informações. Altere a propriedade *Caption* do botão para *Logar* (veja o resultado na **Figura 4**). Dê um duplo *click* no *Button* e no seu evento *OnClick* adicione o código da **Listagem 7**.

Veja que apenas realizamos uma pequena comparação, verificando se nosso *txtLogin* é igual a *Rafael* e *txtSenha* é igual a *123*. Com isso, sendo verdadeiro ele informa que a propriedade *ActiveLayer* de nosso *Panel* recebe 1. Como o seu valor *Defaut* é 0, no momento que setamos para 1, nosso sistema de *login* não será exibido e com isso podemos configurar nosso *Panel* para exibir no *Layer 1* uma tela de sucesso do *Login*.

Para isso, com o *Panel* (*PDados*) selecionado, altere sua propriedade *ActiveLayer* para 1. Logo após acrescente um *Label* e altere seu *Caption* para LOGADO. Em seguida defina novamente a mesma como 0, com isso a *Layer default* de nosso *Panel*

Listagem 7. Evento OnClick do Button

```
function Button1Click($sender, $params)
        {
          if($this->txtLogin->Text == "rafael" && $this->
            txtSenha->Text == "123"){
              $this->PDados->ActiveLayer = 1;
        }
}
```

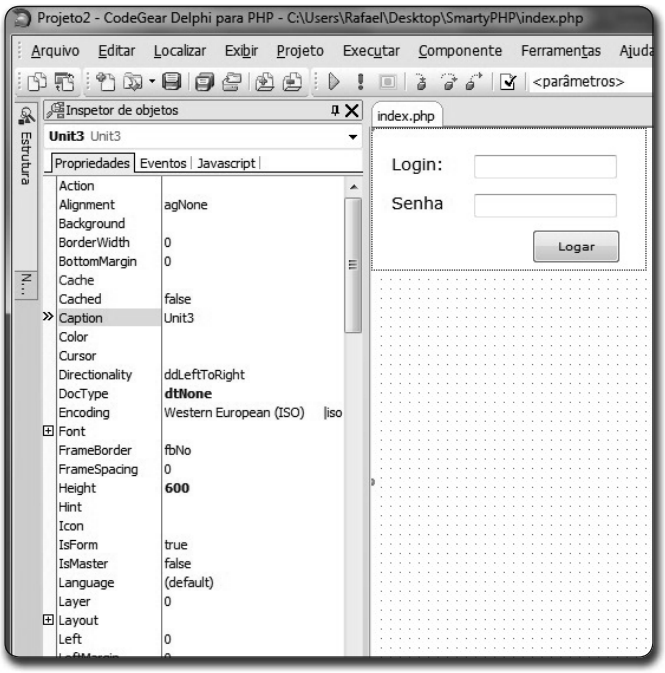


Figura 4. Tela de Login

será a tela de *login*. Veja que já temos toda estrutura definida na parte *PHP*, agora vamos partir para o *template*.

Definindo o template no Delphi for PHP

Crie um novo documento *HTML* padrão, você pode utilizar qualquer editor, até mesmo um bloco de notas. Para você trabalhar com a *Smarty Template* no *Delphi for PHP*, é preciso que sejam inseridas algumas *tags* em seus *templates*, são elas: *HeaderCode*, *StartForm* e *EndForm*. Veja na **Listagem 8** um exemplo de *template* pronto para ser utilizado no *Delphi For PHP*. Basicamente temos um documento *HTML* padrão com as tags citadas anteriormente.

Note que entre as tags *StartForm* e *EndForm* você pode acrescentar as *tags* de seus elementos do *Delphi for PHP*, utilizando *{%$tag%}*. No nosso caso vamos simplesmente inserir nosso *PDados*. Estamos a um passo de finalizar. Para isso com seu projeto aberto no *D4PHP*, altere as propriedades *TemplateEngine* do formulário para *SmartyTemplate* e *TemplateFilename* indicando seu *template HTML* que criamos anteriormente. Com isso já somos capazes de executar nossa aplicação e testá-la. Na **Listagem 9** você confere o código completo do *index.php* do projeto.

Conclusão

Apresentamos nesse artigo a biblioteca *Smarty Template*, uma biblioteca de *templates* para PHP. Ela vai muito mais além do que citamos, utilizando seus padrões e recursos seria possível até mesmo utilizar o padrão MVC. Também criamos um pequeno exemplo utilizando a ferramenta *Delphi for PHP*, que permite usar o recurso com uma facilidade e produtividade muito grande na implementação, e um resultado eficaz. ●

Listagem 8. Exemplo de template

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
{%$HeaderCode%}
<title>Untitled Document</title>
</head>

<body>
{%$StartForm%}

{%$EndForm%}
</body>
</html>
```

Listagem 9. Exemplo completo - index.php

```
<?php
require_once("vcl/vcl.inc.php");
//Inclusões
use_unit("forms.inc.php");
use_unit("extctrls.inc.php");
use_unit("stdctrls.inc.php");

//Definição de classe
class Unit3 extends Page
{
        public $Label3 = null;
        public $Button1 = null;
        public $txtLogin = null;
        public $txtSenha = null;
        public $Label2 = null;
        public $Label1 = null;
        public $PDados = null;
        function Button1Click($sender, $params)
        {
          if($this->txtLogin->Text == "rafael" && $this->
           txtSenha->Text == "123")
              $this->PDados->ActiveLayer = 1;
        }

}

global $application;

global $Unit3;

//Cria o formulário
$Unit3=new Unit3($application);

//Ler do arquivo de recursos
$Unit3->loadResource(__FILE__);

//Mostrar o formulário
$Unit3->show();

?>
```

Links

Download Completo da biblioteca Smarty Template

http://www.smarty.net

Wamp Server – Pacote que contém Apache, PHP e MySQL

http://www.wampserver.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Existem momentos onde a estrutura faz toda a diferença.

Se o tempo é ruim, você encontra na Alog um porto seguro. Isto porque sua empresa reduz custos e melhora a governança em TI. Entre em contato e descubra porque a Alog é líder em hosting gerenciado.

REDUÇÃO DOS CUSTOS AUMENTO DA GOVERNANÇA EM TI

0800 282 3330 | alog.com.br

ALOG
DATA CENTERS DO BRASIL

HOSTING GERENCIADO | COLOCATION | CONTINGÊNCIA | E-MAIL CORPORATIVO | CONECTIVIDADE | SERVIÇOS PROFISSIONAIS

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.